Text Analytics Toolbox™

Reference

MATLAB®

R2021a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Text Analytics Toolbox™ Reference

Revision History

September 2017	Online Only	New for Version 1.0
March 2018	Online Only	Revised for Version 1.1 (Release 2018a)
September 2018	Online Only	Revised for Version 1.2 (Release 2018b)
March 2019	Online Only	Revised for Version 1.3 (Release 2019a)
September 2019	Online Only	Revised for Version 1.4 (Release 2019b)
March 2020	Online Only	Revised for Version 1.5 (Release 2020a)
September 2020	Online Only	Revised for Version 1.6 (Release 2020b)
March 2021	Online Only	Revised for Version 1.7 (Release 2021a)

Contents

Functions

abbreviations

Table of common abbreviations

Syntax

```
tbl = abbreviations
tbl = abbreviations('Language',language)
```

Description

Abbreviations containing periods like "appt.", "Dr.", and "fig." affect sentence detection. The `addSentenceDetails` and `addPartOfSpeechDetails` functions use tables of abbreviations to detect sentence boundaries. The `abbreviations` function outputs the default table used by these functions. You can use this table to help create custom tables of abbreviations to specify sentence detection behavior.

The function supports English, Japanese, German, and Korean language. The Japanese and Korean abbreviation lists are empty because in these languages, abbreviations do not usually impact sentence detection.

`tbl = abbreviations` returns a table of common English abbreviations.

`tbl = abbreviations('Language',language)` specifies the abbreviation language.

Examples

Table of Abbreviations

View a table of abbreviations. You can use this table to detect abbreviations and sentences when using `addSentenceDetails`.

```
tbl = abbreviations;
head(tbl)
```

```
ans=8×2 table
    Abbreviation     Usage 
    ____________    _______

       "ATS"        regular
       "Ao"         regular
       "BEF"        regular
       "Ba"         regular
       "Bd"         regular
       "Bi"         regular
       "Bq"         regular
       "Cent"       regular
```

Table of German Abbreviations

View a table of German abbreviations. Use this table to help create custom tables of abbreviations for sentence detection when using `addSentenceDetails`.

```
tbl = abbreviations('Language','de');
head(tbl)
```

```
ans=8×2 table
    Abbreviation     Usage 
    ____________    _______

       "A.T"        regular
       "ABl"        regular
       "Abb"        regular
       "Abdr"       regular
       "Abf"        regular
       "Abfl"       regular
       "Abh"        regular
       "Abk"        regular
```

Input Arguments

`language` — Abbreviation language
`'en'` (default) | `'ja'` | `'de'` | `'ko'`

Abbreviation language, specified as one of the following:

- `'en'` – English
- `'ja'` – Japanese
- `'de'` – German
- `'ko'` – Korean

If you specify `'ja'` or `'ko'`, then the function returns an empty table. For more information about language support in Text Analytics Toolbox™, see “Language Considerations”.

Output Arguments

`tbl` — Table of abbreviations
table

Table of abbreviations. The `addSentenceDetails` and `splitSentences` functions, by default, use this table to detect sentence boundaries. This table only contains abbreviations typically written with periods.

The table has two variables:

- `Abbreviation` – Abbreviation, specified as a string
- `Usage` – Type of abbreviation, specified as a categorical scalar

The following table describes the possible values of `Usage` and the behavior of `addSentenceDetails` and `splitSentences` when observing abbreviations of these types.

Usage	Behavior	Example Abbreviation	Example Text	Detected Sentences

`regular`	If the next word is a capitalized sentence starter, then break at the trailing period. Otherwise, do not break at the trailing period.	"appt."	`"Book an appt. We'll meet then."`	`"Book an appt."`
				
				`"We'll meet then."`

			`"Book an appt. today."`	`"Book an appt. today."`
`inner`	Do not break after trailing period.	"Dr."	`"Dr. Smith."`	`"Dr. Smith."`
`reference`	If the next token is not a number, then break at a trailing period. If the next token is a number, then do not break at the trailing period.	"fig."	`"See fig. 3."`	`"See fig. 3."`

			`"Try a fig. They are nice."`	`"Try a fig."`
				
				`"They are nice."`

`unit`	If the previous word is a number and the following word is a capitalized sentence starter, then break at a trailing period.	"in."	`"The height is 30 in. The width is 10 in."`	`"The height is 30 in."`
				
				`"The width is 10 in."`

	If the previous word is a number and the following word is not capitalized, then do not break at a trailing period.		`"The item is 10 in. wide."`	`"The item is 10 in. wide."`

	If the previous word is not a number, then break at a trailing period.		`"Come in. Sit down."`	`"Come in."`
				
				`"Sit down."`

The Japanese and Korean abbreviation lists are empty because in these languages, abbreviations do not usually impact sentence detection

See Also

`addPartOfSpeechDetails` | `addSentenceDetails` | `tokenDetails` | `tokenizedDocument`

Topics

"Prepare Text Data for Analysis"

"Create Simple Text Model for Classification"
"Language Considerations"

Introduced in R2018a

addDocument

Add documents to bag-of-words or bag-of-n-grams model

Syntax

```
newBag = addDocument(bag,documents)
```

Description

`newBag = addDocument(bag,documents)` adds `documents` to the bag-of-words or bag-of-n-grams model `bag`.

Examples

Add Documents to Bag-of-Words Model

Create a bag-of-words model from an array of tokenized documents.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [2x7 double]
      Vocabulary: [1x7 string]
        NumWords: 7
    NumDocuments: 2
```

Create another array of tokenized documents and add it to the same bag-of-words model.

```
documents = tokenizedDocument([
    "a third example of a short sentence"
    "another short sentence"]);
newBag = addDocument(bag,documents)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [4x9 double]
      Vocabulary: [1x9 string]
        NumWords: 9
    NumDocuments: 4
```

Import Text from Multiple Files Using a File Datastore

If your text data is contained in multiple files in a folder, then you can import the text data into MATLAB using a file datastore.

Create a file datastore for the example sonnet text files. The examples sonnets have file names "`exampleSonnetN.txt`", where `N` is the number of the sonnet. Specify the read function to be `extractFileText`.

```
readFcn = @extractFileText;
fds = fileDatastore('exampleSonnet*.txt','ReadFcn',readFcn)
```

```
fds = 
  FileDatastore with properties:

                       Files: {
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonne
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonne
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonne
                               ... and 1 more
                              }
                     Folders: {
                              ' ...\Bdoc21a_1606923_5032\ib8F3FCD\26\tp5980089c\textanalytics-ex
                              }
                 UniformRead: 0
                    ReadMode: 'file'
                   BlockSize: Inf
                  PreviewFcn: @extractFileText
      SupportedOutputFormats: [1x16 string]
                     ReadFcn: @extractFileText
    AlternateFileSystemRoots: {}
```

Create an empty bag-of-words model.

```
bag = bagOfWords
```

```
bag = 
  bagOfWords with properties:

          Counts: []
      Vocabulary: [1x0 string]
        NumWords: 0
    NumDocuments: 0
```

Loop over the files in the datastore and read each file. Tokenize the text in each file and add the document to `bag`.

```
while hasdata(fds)
    str = read(fds);
    document = tokenizedDocument(str);
    bag = addDocument(bag,document);
end
```

View the updated bag-of-words model.

```
bag
```

```
bag = 
  bagOfWords with properties:

          Counts: [4x276 double]
      Vocabulary: [1x276 string]
        NumWords: 276
    NumDocuments: 4
```

Input Arguments

bag — Input bag-of-words or bag-of-n-grams model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object.

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

Output Arguments

newBag — Output model
`bagOfWords` object | `bagOfNgrams` object

Output model, returned as a `bagOfWords` object or a `bagOfNgrams` object. The type of `newBag` is the same as the type of `bag`.

See Also

`bagOfNgrams` | `bagOfWords` | `removeDocument` | `removeEmptyDocuments` | `tokenizedDocument`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Analyze Text Data Using Topic Models”
“Analyze Text Data Using Multiword Phrases”
“Visualize Text Data Using Word Clouds”
“Classify Text Data Using Deep Learning”

Introduced in R2017b

addEntityDetails

Add entity tags to documents

Syntax

```
updatedDocuments = addEntityDetails(documents)
updatedDocuments = addEntityDetails(documents,Name,Value)
```

Description

Use `addEntityDetails` to add entity tags to documents.

Use `addEntityDetails` to detect person names, locations, organizations, and other named entities in text. This process is known as *named entity recognition*.

The function supports English, Japanese, German, and Korean text.

`updatedDocuments = addEntityDetails(documents)` detects the named entities in `documents`. The function adds details to the tokens with missing entity details only. To get the entity details from `updatedDocuments`, use `tokenDetails`.

`updatedDocuments = addEntityDetails(documents,Name,Value)` also specifies additional options using one or more name-value pairs.

Tip Use `addEntityDetails` before using the `lower`, `upper`, `normalizeWords`, `removeWords`, and `removeStopWords` functions as `addEntityDetails` uses information that is removed by these functions.

Examples

Add Named Entity Tags to Documents

Create a tokenized document array.

```
str = [
    "Mary moved to Natick, Massachusetts."
    "John uses MATLAB at MathWorks."];
documents = tokenizedDocument(str);
```

Add the entity details to the documents using the `addEntityDetails` function. This function detects the named entities in the text and adds the details to the table returned by the `tokenDetails` function. View the updated token details of the first few tokens.

```
documents = addEntityDetails(documents);
tdetails = tokenDetails(documents)
```

```
tdetails=13×8 table
         Token          DocumentNumber    SentenceNumber    LineNumber       Type       Language
    _______________    ______________    ______________    __________    ___________    ________
    "Mary"                 1                  1                1          letters          en
    "moved"                1                  1                1          letters          en
    "to"                   1                  1                1          letters          en
    "Natick"               1                  1                1          letters          en
    ","                    1                  1                1          punctuation      en
    "Massachusetts"        1                  1                1          letters          en
    "."                    1                  1                1          punctuation      en
    "John"                 2                  1                1          letters          en
    "uses"                 2                  1                1          letters          en
    "MATLAB"               2                  1                1          letters          en
    "at"                   2                  1                1          letters          en
    "MathWorks"            2                  1                1          letters          en
    "."                    2                  1                1          punctuation      en
```

View the words tagged with the entities `"person"`, `"location"`, `"organization"`, or `"other"`. These words are the words not tagged with `"non-entity"`.

```
idx = tdetails.Entity ~= "non-entity";
tdetails.Token(idx)
```

```
ans = 6x1 string
    "Mary"
    "Natick"
    "Massachusetts"
    "John"
    "MATLAB"
    "MathWorks"
```

Add Named Entity Tags to Japanese Text

Tokenize Japanese text using `tokenizedDocument`.

```
str = [
    "マリーさんはボストンからニューヨークに引っ越しました。"
    "駅で鈴木さんに迎えに行きます。"
    "東京は大阪より大きいですか？"
    "東京に行った時、新宿や渋谷などいろいろな所を訪れました。"];
documents = tokenizedDocument(str);
```

For Japanese text, the software automatically adds named entity tags, so you do not need to use the `addEntityDetails` function. This software detects person names, locations, organizations, and other named entities. To view the entity details, use the `tokenDetails` function.

```
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×8 table
       Token          DocumentNumber    LineNumber     Type       Language    PartOfSpeech       Le
    ____________      ______________    __________    _______     ________    ____________     ______

    "マリー"                1                1          letters       ja        proper-noun      "マリ
    "さん"                 1                1         letters       ja        noun             "さん"
    "は"                   1                1         letters       ja        adposition       "は"
    "ボストン"            1              1         letters       ja       proper-noun      "ボスト
    "から"               1              1         letters       ja       adposition       "から"
    "ニューヨーク"          1              1         letters       ja       proper-noun      "ニュー
    "に"                 1              1         letters       ja       adposition       "に"
    "引っ越し"             1              1         letters       ja       verb             "引っ越
```

View the words tagged with entity `"person"`, `"location"`, `"organization"`, or `"other"`. These words are the words not tagged `"non-entity"`.

```
idx = tdetails.Entity ~= "non-entity";
tdetails(idx,:).Token
```

```
ans = 11x1 string
    "マリー"
    "さん"
    "ボストン"
    "ニューヨーク"
    "鈴木"
    "さん"
    "東京"
    "大阪"
    "東京"
    "新宿"
    "渋谷"
```

Add Named Entity Tags to German Text

Tokenize German text using `tokenizedDocument`.

```
str = [
    "Ernst zog von Frankfurt nach Berlin."
    "Besuchen Sie Volkswagen in Wolfsburg."];
documents = tokenizedDocument(str);
```

To add entity tags to German text, use the `addEntityDetails` function. This function detects person names, locations, organizations, and other named entities.

```
documents = addEntityDetails(documents);
```

To view the entity details, use the `tokenDetails` function.

```
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×8 table
       Token        DocumentNumber    SentenceNumber    LineNumber      Type       Language
    ___________    ______________    ______________    __________    __________    ________

    "Ernst"              1                 1               1          letters         de
    "zog"                1                 1               1          letters         de
    "von"                1                 1               1          letters         de
    "Frankfurt"          1                 1               1          letters         de
    "nach"               1                 1               1          letters         de
    "Berlin"             1                 1               1          letters         de
    "."                 1                 1               1         punctuation       de        p
    "Besuchen"          2                 1               1         letters           de        v
```

View the words tagged with entity `"person"`, `"location"`, `"organization"`, or `"other"`. These words are the words not tagged with `"non-entity"`.

```
idx = tdetails.Entity ~= "non-entity";
tdetails(idx,:)
```

```
ans=5×8 table
       Token        DocumentNumber    SentenceNumber    LineNumber     Type      Language    Par
    ____________    ______________    ______________    __________    _______    ________    ___

    "Ernst"               1                 1               1         letters       de       prop
    "Frankfurt"           1                 1               1         letters       de       prop
    "Berlin"              1                 1               1         letters       de       prop
    "Volkswagen"          2                 1               1         letters       de       nour
    "Wolfsburg"           2                 1               1         letters       de       prop
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DiscardKnownValues',true` specifies to discard previously computed details and recompute them.

`RetokenizeMethod` — Method to retokenize documents
`'entity'` (default) | `'none'`

Method to retokenize documents, specified as one of the following:

- `'entity'` – Transform the tokens for named entity recognition. The function merges tokens from the same entity into a single token.
- `'none'` – Do not retokenize the documents.

`DiscardKnownValues` — Option to discard previously computed details
`false` (default) | `true`

Option to discard previously computed details and recompute them, specified as `true` or `false`.

Data Types: `logical`

Output Arguments

`updatedDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array. To get the token details from `updatedDocuments`, use `tokenDetails`.

Algorithms

Language Details

`tokenizedDocument` objects contain details about the tokens including language details. The language details of the input documents determine the behavior of `addEntityDetails`. The `tokenizedDocument` function, by default, automatically detects the language of the input text. To specify the language details manually, use the `'Language'` name-value pair argument of `tokenizedDocument`. To view the token details, use the `tokenDetails` function.

See Also

`abbreviations` | `addLanguageDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `addSentenceDetails` | `addTypeDetails` | `corpusLanguage` | `splitSentences` | `tokenDetails` | `tokenizedDocument` | `topLevelDomains`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Visualize Text Data Using Word Clouds"
"Language Considerations"
"Japanese Language Support"
"German Language Support"

Introduced in R2019a

addLanguageDetails

Add language identifiers to documents

Syntax

```
updatedDocuments = addLanguageDetails(documents)
updatedDocuments = addLanguageDetails(documents,Name,Value)
```

Description

Use `addLanguageDetails` to add language identifiers to documents.

The function supports English, Japanese, German, and Korean text.

`updatedDocuments = addLanguageDetails(documents)` detects the language of `documents` and updates the token details. The function adds details to the tokens with missing language details only. To get the language details from `updatedDocuments`, use `tokenDetails`.

`updatedDocuments = addLanguageDetails(documents,Name,Value)` specifies additional options using one or more name-value pairs.

Tip Use `addLanguageDetails` before using the `lower` and `upper` functions as `addLanguageDetails` uses information that is removed by this functions.

Examples

Add Language Details to Documents

Manually tokenize some text by splitting it into an array of words. Convert the manually tokenized text into a `tokenizedDocument` object by setting the `'TokenizeMethod'` option to `'none'`.

```
str = split("an example of a short sentence")';
documents = tokenizedDocument(str,'TokenizeMethod','none');
```

View the token details using `tokenDetails`.

```
tdetails = tokenDetails(documents)
```

```
tdetails=6×2 table
      Token       DocumentNumber
    __________    ______________

    "an"                1
    "example"           1
    "of"                1
    "a"                 1
    "short"             1
    "sentence"          1
```

When you specify `'TokenizeMethod','none'`, the function does not automatically detect the language details of the documents. To add the language details, use the `addLanguageDetails` function. This function, by default, automatically detects the language.

```
documents = addLanguageDetails(documents);
```

View the updated token details using `tokenDetails`.

```
tdetails = tokenDetails(documents)
```

```
tdetails=6×4 table
      Token       DocumentNumber     Type      Language
    __________    ______________    _______    ________

    "an"                1           letters       en
    "example"           1           letters       en
    "of"                1           letters       en
    "a"                 1           letters       en
    "short"             1           letters       en
    "sentence"          1           letters       en
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DiscardKnownValues',true` specifies to discard previously computed details and recompute them.

Language — Language
`'en'` | `'ja'` | `'de'` | `'ko'`

Language, specified as one of the following:

- `'en'` – English
- `'ja'` – Japanese
- `'de'` – German
- `'ko'` – Korean

If you do not specify a value, then the function detects the language from the input text using the `corpusLanguage` function.

This option specifies the language details of the tokens. To view the language details of the tokens, use `tokenDetails`. These language details determine the behavior of the `removeStopWords`, `addPartOfSpeechDetails`, `normalizeWords`, `addSentenceDetails`, and `addEntityDetails` functions on the tokens.

For more information about language support in Text Analytics Toolbox, see “Language Considerations”.

`DiscardKnownValues` — Option to discard previously computed details
`false` (default) | `true`

Option to discard previously computed details and recompute them, specified as `true` or `false`.

Data Types: `logical`

Output Arguments

`updatedDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array. To get the token details from `updatedDocuments`, use `tokenDetails`.

See Also

`abbreviations` | `addEntityDetails` | `addEntityDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `addSentenceDetails` | `addTypeDetails` | `corpusLanguage` | `splitSentences` | `tokenDetails` | `tokenizedDocument` | `topLevelDomains`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Visualize Text Data Using Word Clouds”
“Japanese Language Support”
“Language Considerations”
“German Language Support”

Introduced in R2018b

addLemmaDetails

Add lemma forms of tokens to documents

Syntax

```
updatedDocuments = addLemmaDetails(documents)
updatedDocuments = addLemmaDetails(documents,'DiscardKnownValues',true)
```

Description

Use `addLemmaDetails` to add lemma forms to documents.

The function supports English, Japanese, and Korean text.

`updatedDocuments = addLemmaDetails(documents)` adds lemma details to `documents` and updates the token details. To get the lemma details from `updatedDocuments`, use `tokenDetails`.

`updatedDocuments = addLemmaDetails(documents,'DiscardKnownValues',true)` discards previously computed details and recomputes them.

Tip Use `addLemmaDetails` before using the `lower`, `upper`, and `normalizeWords` functions as `addLemmaDetails` uses information that is removed by these functions.

Examples

Add Lemma Details to Documents

Create a tokenized document array.

```
str = [ ...
    "The dogs ran after the cat."
    "I am building a house."];
documents = tokenizedDocument(str);
```

Add lemma details to the documents using `addLemmaDetails`. This function lemmatizes the text and adds the lemma form of each token to the table returned by `tokenDetails`. View the updated token details of the first few tokens.

```
documents = addLemmaDetails(documents);
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×6 table
     Token     DocumentNumber    LineNumber      Type       Language     Lemma
    _______    ______________    __________    _________    ________    _______

    "The"            1               1         letters         en       "the"
    "dogs"           1               1         letters         en       "dog"
    "ran"            1               1         letters         en       "run"
    "after"          1                1         letters          en       "after"
    "the"            1                1         letters          en       "the"
    "cat"            1                1         letters          en       "cat"
    "."              1                1         punctuation      en       "."
    "I"              2                1         letters          en       "i"
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`updatedDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array. To get the token details from `updatedDocuments`, use `tokenDetails`.

See Also

`addEntityDetails` | `addLanguageDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `addSentenceDetails` | `addTypeDetails` | `normalizeWords` | `tokenDetails` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Language Considerations"

Introduced in R2018b

addPartOfSpeechDetails

Add part-of-speech tags to documents

Syntax

```
updatedDocuments = addPartOfSpeechDetails(documents)
updatedDocuments = addPartOfSpeechDetails(documents,Name,Value)
```

Description

Use `addPartOfSpeechDetails` to add part-of-speech tags to documents.

The function supports English, Japanese, German, and Korean text.

`updatedDocuments = addPartOfSpeechDetails(documents)` detects parts of speech in `documents` and updates the token details. The function, by default, retokenizes the text for part-of-speech tagging. For example, the function splits the word "you're" into the tokens "you" and "'re". To get the part-of-speech details from `updatedDocuments`, use `tokenDetails`.

`updatedDocuments = addPartOfSpeechDetails(documents,Name,Value)` specifies additional options using one or more name-value pair arguments.

Tip Use `addPartOfSpeechDetails` before using the `lower`, `upper`, `erasePunctuation`, `normalizeWords`, `removeWords`, and `removeStopWords` functions as `addPartOfSpeechDetails` uses information that is removed by these functions.

Examples

Add Part-of-Speech Details to Documents

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

View the token details of the first few tokens.

```
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×5 table
       Token       DocumentNumber    LineNumber     Type      Language
    ___________    ______________    __________    _______    ________
    "fairest"           1               1         letters       en
    "creatures"         1               1         letters       en
    "desire"            1               1         letters       en
    "increase"          1               1         letters       en
    "thereby"           1               1         letters       en
    "beautys"           1               1         letters       en
    "rose"              1               1         letters       en
    "might"             1               1         letters       en
```

Add part-of-speech details to the documents using the `addPartOfSpeechDetails` function. This function first adds sentence information to the documents, and then adds the part-of-speech tags to the table returned by `tokenDetails`. View the updated token details of the first few tokens.

```
documents = addPartOfSpeechDetails(documents);
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×7 table
       Token        DocumentNumber    SentenceNumber    LineNumber     Type       Language     Par
    ___________    ______________    ______________    __________    _______    ________    ____

    "fairest"           1                 1                1         letters       en       adje
    "creatures"         1                 1                1         letters       en       noun
    "desire"            1                 1                1         letters       en       noun
    "increase"          1                 1                1         letters       en       noun
    "thereby"           1                 1                1         letters       en       adve
    "beautys"           1                 1                1         letters       en       noun
    "rose"              1                 1                1         letters       en       noun
    "might"             1                 1                1         letters       en       auxi
```

Get Part of Speech Details of Japanese Text

Tokenize Japanese text using `tokenizedDocument`.

```
str = [
    "恋に悩み、苦しむ。"
    "恋の悩みで 苦しむ。"
    "空に星が輝き、瞬いている。"
    "空の星が輝きを増している。"
    "駅までは遠くて、歩けない。"
    "遠くの駅まで歩けない。"
    "すもももももももものうち。"];
documents = tokenizedDocument(str);
```

For Japanese text, you can get the part-of-speech details using `tokenDetails`. For English text, you must first use `addPartOfSpeechDetails`.

```
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×8 table
     Token     DocumentNumber    LineNumber       Type        Language    PartOfSpeech     Lemma
    _______    ______________    __________    ___________    ________    ____________    ______
    "恋"            1               1         letters          ja       noun            "恋"
    "に"            1               1         letters          ja       adposition      "に"
    "悩み"          1               1         letters          ja       verb            "悩む"
    "、"            1               1         punctuation      ja       punctuation     "、"
    "苦しむ"        1               1         letters          ja       verb            "苦しむ
    "。"            1               1         punctuation      ja       punctuation     "。"
    "恋"            2               1         letters          ja       noun            "恋"
    "の"            2               1         letters          ja       adposition      "の"
```

Get Part of Speech Details of German Text

Tokenize German text using `tokenizedDocument`.

```
str = [
    "Guten Morgen. Wie geht es dir?"
    "Heute wird ein guter Tag."];
documents = tokenizedDocument(str)
```

```
documents = 
  2x1 tokenizedDocument:

    8 tokens: Guten Morgen . Wie geht es dir ?
    6 tokens: Heute wird ein guter Tag .
```

To get the part of speech details for German text, first use `addPartOfSpeechDetails`.

```
documents = addPartOfSpeechDetails(documents);
```

To view the part of speech details, use the `tokenDetails` function.

```
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×7 table
     Token       DocumentNumber    SentenceNumber    LineNumber       Type        Language    Par
    ________    ______________    ______________    __________    ___________    ________    ___

    "Guten"            1                 1               1        letters           de       adje
    "Morgen"           1                 1               1        letters           de       nou
    "."                1                 1               1        punctuation       de       punc
    "Wie"              1                 2               1        letters           de       adve
    "geht"             1                 2               1        letters           de       verb
    "es"               1                 2               1        letters           de       pron
    "dir"              1                 2               1        letters           de       pron
    "?"                1                 2               1        punctuation       de       punc
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DiscardKnownValues',true` specifies to discard previously computed details and recompute them.

`RetokenizeMethod` — Method to retokenize documents

`'part-of-speech'` (default) | `'none'`

Method to retokenize documents, specified as one of the following:

- `'part-of-speech'` – Transform the tokens for part-of-speech tagging. The function performs these tasks:
  - Split compound words. For example, split the compound word `"wanna"` into the tokens `"want"` and `"to"`. This includes compound words containing apostrophes. For example, the function splits the word `"don't"` into the tokens `"do"` and `"n't"`.
  - Merge periods that do not end sentences with preceding tokens. For example, merge the tokens `"Mr"` and `"."` into the token `"Mr."`.
  - For German text, merge abbreviations that span multiple tokens. For example, merge the tokens `"z"`, `"."`, `"B"`, and `"."` into the single token `"z. B."`.
  - Merge runs of periods into ellipses. For example, merge three instances of `"."` into the single token `"..."`.
- `'none'` – Do not retokenize the documents.

`Abbreviations` — List of abbreviations

string array | character vector | cell array of character vectors | table

List of abbreviations for sentence detection, specified as a string array, character vector, cell array of character vectors, or a table.

If the input documents do not contain sentence details, then the function first runs the `addSentenceDetails` function and specifies the abbreviation list given by `'Abbreviations'`. To specify more options for sentence detection (for example, sentence starters) use the `addSentenceDetails` function before using `addPartOfSpeechDetails` details.

If `Abbreviations` is a string array, character vector, or cell array of character vectors, then the function treats these as regular abbreviations. If the next word is a capitalized sentence starter, then the function breaks at the trailing period. The function ignores any differences in the letter case of the abbreviations. Specify the sentence starters using the `Starters` name-value pair.

To specify different behaviors when splitting sentences at abbreviations, specify `Abbreviations` as a table. The table must have variables named `Abbreviation` and `Usage`, where `Abbreviation` contains the abbreviations, and `Usage` contains the type of each abbreviation. The following table describes the possible values of `Usage`, and the behavior of the function when passed abbreviations of these types.

Usage	Behavior	Example Abbreviation	Example Text	Detected Sentences
`regular`	If the next word is a capitalized sentence starter, then break at the trailing period. Otherwise, do not break at the trailing period.	"appt."	`"Book an appt. We'll meet then."`	`"Book an appt."` `"We'll meet then."`
			`"Book an appt. today."`	`"Book an appt. today."`
`inner`	Do not break after trailing period.	"Dr."	`"Dr. Smith."`	`"Dr. Smith."`
`reference`	If the next token is not a number, then break at a trailing period. If the next token is a number, then do not break at the trailing period.	"fig."	`"See fig. 3."`	`"See fig. 3."`
			`"Try a fig. They are nice."`	`"Try a fig."` `"They are nice."`
`unit`	If the previous word is a number and the following word is a capitalized sentence starter, then break at a trailing period.	"in."	`"The height is 30 in. The width is 10 in."`	`"The height is 30 in."` `"The width is 10 in."`
	If the previous word is a number and the following word is not capitalized, then do not break at a trailing period.		`"The item is 10 in. wide."`	`"The item is 10 in. wide."`
	If the previous word is not a number, then break at a trailing period.		`"Come in. Sit down."`	`"Come in."` `"Sit down."`

The default value is the output of the `abbreviations` function. For Japanese and Korean text, abbreviations do not usually impact sentence detection.

Tip By default, the function treats single letter abbreviations, such as "V.", or tokens with mixed single letters and periods, such as "U.S.A." as regular abbreviations. You do not need to include these abbreviations in `Abbreviations`.

Data Types: `char` | `string` | `table` | `cell`

`DiscardKnownValues` — Option to discard previously computed details
`false` (default) | `true`

Option to discard previously computed details and recompute them, specified as `true` or `false`.

Data Types: `logical`

Output Arguments

`updatedDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array. To get the token details from `updatedDocuments`, use `tokenDetails`.

More About

Part-of-Speech Tags

The `addPartOfSpeechDetails` function adds part-of-speech tags to the table returned by the `tokenDetails` function. The function tags each token with a categorical tag with one of the following class names:

- `"adjective"` – Adjective
- `"adposition"` – Adposition
- `"adverb"` – Adverb
- `"auxiliary-verb"` – Auxiliary verb
- `"coord-conjunction"` – Coordinating conjunction
- `"determiner"` – Determiner
- `"interjection"` – Interjection
- `"noun"` – Noun
- `"numeral"` – Numeral
- `"particle"` – Particle
- `"pronoun"` – Pronoun
- `"proper-noun"` – Proper noun
- `"punctuation"` – Punctuation
- `"subord-conjunction"` – Subordinating conjucntion
- `"symbol"` – Symbol
- `"verb"` – Verb
- `"other"` – Other

Algorithms

If the input documents do not contain sentence details, then the function first runs `addSentenceDetails`.

See Also

`addEntityDetails` | `addLanguageDetails` | `addLemmaDetails` | `addSentenceDetails` | `addTypeDetails` | `normalizeWords` | `tokenDetails` | `tokenizedDocument`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Language Considerations"
"Japanese Language Support"
"German Language Support"

Introduced in R2018b

addSentenceDetails

Add sentence numbers to documents

Syntax

```
updatedDocuments = addSentenceDetails(documents)
updatedDocuments = addSentenceDetails(documents,Name,Value)
```

Description

Use `addSentenceDetails` to add sentence information to documents.

The function supports English, Japanese, German, and Korean text.

`updatedDocuments = addSentenceDetails(documents)` detects the sentence boundaries in `documents` and updates the token details. To get the sentence details from `updatedDocuments`, use `tokenDetails`.

`updatedDocuments = addSentenceDetails(documents,Name,Value)` specifies additional options using one or more name-value pair arguments.

Tip Use `addSentenceDetails` before using the `lower`, `upper`, `erasePunctuation`, `normalizeWords`, `removeWords`, and `removeStopWords` functions as `addSentenceDetails` uses information that is removed by these functions.

Examples

Add Sentence Details to Documents

Create a tokenized document array.

```
str = [ ...
    "This is an example document. It has two sentences."
    "This document has one sentence."
    "Here is another example document. It also has two sentences."];
documents = tokenizedDocument(str);
```

Add sentence details to the documents using `addSentenceDetails`. This function adds the sentence numbers to the table returned by `tokenDetails`. View the updated token details of the first few tokens.

```
documents = addSentenceDetails(documents);
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×6 table
      Token        DocumentNumber    SentenceNumber    LineNumber       Type       Language
    __________    ______________    ______________    __________    ___________    ________
    "This"              1                  1                1         letters           en
    "is"                1                  1                1         letters           en
    "an"                1                  1                1         letters           en
    "example"           1                  1                1         letters           en
    "document"          1                  1                1         letters           en
    "."                 1                  1                1         punctuation       en
    "It"                1                  2                1         letters           en
    "has"               1                  2                1         letters           en
```

View the token details of the second sentence of the third document.

```
idx = tdetails.DocumentNumber == 3 & ...
    tdetails.SentenceNumber == 2;
tdetails(idx,:)
```

```
ans=6×6 table
       Token        DocumentNumber    SentenceNumber    LineNumber       Type        Language
    ___________     ______________    ______________    __________    ___________    ________

    "It"                  3                 2               1         letters           en   
    "also"                3                 2               1         letters           en   
    "has"                 3                 2               1         letters           en   
    "two"                 3                 2               1         letters           en   
    "sentences"           3                 2               1         letters           en   
    "."                   3                 2               1         punctuation       en   
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Abbreviations',["cm" "mm" "in"]` specifies to detect sentences boundaries where these abbreviations are followed by a period and a capitalized sentence starter.

`Abbreviations` — List of abbreviations
string array | character vector | cell array of character vectors | table

List of abbreviations, specified as a string array, character vector, cell array of character vectors, or a table.

If `Abbreviations` is a string array, character vector, or cell array of character vectors, then the function treats these as regular abbreviations. If the next word is a capitalized sentence starter, then the function breaks at the trailing period. The function ignores any differences in the letter case of the abbreviations. Specify the sentence starters using the `Starters` name-value pair.

To specify different behaviors when splitting sentences at abbreviations, specify `Abbreviations` as a table. The table must have variables named `Abbreviation` and `Usage`, where `Abbreviation` contains the abbreviations, and `Usage` contains the type of each abbreviation. The following table describes the possible values of `Usage`, and the behavior of the function when passed abbreviations of these types.

Usage	Behavior	Example Abbreviation	Example Text	Detected Sentences

`regular`	If the next word is a capitalized sentence starter, then break at the trailing period. Otherwise, do not break at the trailing period.	"appt."	`"Book an appt. We'll meet then."`	`"Book an appt."`
				
				`"We'll meet then."`

			`"Book an appt. today."`	`"Book an appt. today."`
`inner`	Do not break after trailing period.	"Dr."	`"Dr. Smith."`	`"Dr. Smith."`
`reference`	If the next token is not a number, then break at a trailing period. If the next token is a number, then do not break at the trailing period.	"fig."	`"See fig. 3."`	`"See fig. 3."`

			`"Try a fig. They are nice."`	`"Try a fig."`
				
				`"They are nice."`

`unit`	If the previous word is a number and the following word is a capitalized sentence starter, then break at a trailing period.	"in."	`"The height is 30 in. The width is 10 in."`	`"The height is 30 in."`
				
				`"The width is 10 in."`

	If the previous word is a number and the following word is not capitalized, then do not break at a trailing period.		`"The item is 10 in. wide."`	`"The item is 10 in. wide."`

	If the previous word is not a number, then break at a trailing period.		`"Come in. Sit down."`	`"Come in."`
				
				`"Sit down."`

The default value is the output of the `abbreviations` function. For Japanese and Korean text, abbreviations do not usually impact sentence detection.

Tip By default, the function treats single letter abbreviations, such as "V.", or tokens with mixed single letters and periods, such as "U.S.A." as regular abbreviations. You do not need to include these abbreviations in `Abbreviations`.

Example: `["cm" "mm" "in"]`

Data Types: `char` | `string` | `table` | `cell`

`Starters` — Words that start a sentence
string array | character vector | cell array of character vectors

Words that start a sentence, specified as a string array, character vector, or a cell array of character vectors. If a sentence starter appears capitalized after a regular abbreviation, then the function detects a sentence boundary at the trailing period. The function ignores any differences in the letter case of the sentence starters.

The default value is the output of the `stopWords` function.

Data Types: `char` | `string` | `cell`

`DiscardKnownValues` — Option to discard previously computed details
`false` (default) | `true`

Option to discard previously computed details and recompute them, specified as `true` or `false`.

Data Types: `logical`

Output Arguments

`updatedDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array. To get the token details from `updatedDocuments`, use `tokenDetails`.

More About

Language Considerations

The `addSentenceDetails` function detects sentence boundaries based on punctuation characters and line number information. For English and German text, the function also uses a list of abbreviations passed to the function.

For other languages, you might need to specify your own list of abbreviations for sentence detection. To do this, use the `'Abbreviations'` option of `addSentenceDetails`.

Algorithms

If emoticons or emoji characters appear after a terminating punctuation character, then the function splits the sentence after the emoticons and emoji.

See Also

`abbreviations` | `addEntityDetails` | `addLanguageDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `addTypeDetails` | `splitSentences` | `tokenDetails` | `tokenizedDocument`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Language Considerations"

Introduced in R2018a

addTypeDetails

Add token type details to documents

Syntax

```
updatedDocuments = addTypeDetails(documents)
updatedDocuments = addTypeDetails(documents,Name,Value)
```

Description

`updatedDocuments = addTypeDetails(documents)` detects the token types in `documents` and updates the token details. The function adds type details to the tokens with unknown type only. To get the token types from `updatedDocuments`, use `tokenDetails`.

`updatedDocuments = addTypeDetails(documents,Name,Value)` specifies additional options using one or more name-value pairs.

Tip Use `addTypeDetails` before using the `lower`, `upper`, and `erasePunctuation` functions as `addTypeDetails` uses information that is removed by these functions.

Examples

Add Token Type Details to Documents

Convert manually tokenized text into a `tokenizedDocument` object, setting the `'TokenizeMethod'` option to `'none'`.

```
str = ["For" "more" "information" "," "see" "https://www.mathworks.com" "."];
documents = tokenizedDocument(str,'TokenizeMethod','none')
```

```
documents = 
  tokenizedDocument:

   7 tokens: For more information , see https://www.mathworks.com .
```

View the token details using the `tokenDetails` function.

```
tdetails = tokenDetails(documents)
```

```
tdetails=7×2 table
               Token            DocumentNumber
    ___________________________    ______________

    "For"                              1
    "more"                             1
    "information"                      1
    ","                                1
    "see"                              1
    "https://www.mathworks.com"          1
    "."                                  1
```

If you set `'TokenizeMethod'` to `'none'` in the call to the `tokenizedDocument` function, then it does not detect the types of the tokens. To add the token type details, use the `addTypeDetails` function.

```
documents = addTypeDetails(documents);
```

View the updated token details.

```
tdetails = tokenDetails(documents)
```

```
tdetails=7×3 table
               Token               DocumentNumber       Type    
    ___________________________    ______________    ___________

    "For"                                1           letters    
    "more"                               1           letters    
    "information"                        1           letters    
    ","                                  1           punctuation
    "see"                                1           letters    
    "https://www.mathworks.com"          1           web-address
    "."                                  1           punctuation
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'TopLevelDomains',["com" "net" "org"]` specifies the top-level domains "com", "net", and "org" for web address detection.

`TopLevelDomains` — Top-level domains
character vector | string array | cell array of character vectors

Top-level domains to use for web address detection, specified as a character vector, string array, or cell array of character vectors.

If you do not specify `TopLevelDomains`, then the function uses the output of the `topLevelDomains` function.

Example: `["com" "net" "org"]`

Data Types: `char` | `string` | `cell`

`DiscardKnownValues` — Option to discard previously computed details
`false` (default) | `true`

Option to discard previously computed details and recompute them, specified as `true` or `false`.

Data Types: `logical`

Output Arguments

`updatedDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array. To get the token details from `updatedDocuments`, use `tokenDetails`.

See Also

`abbreviations` | `addEntityDetails` | `addLanguageDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `addSentenceDetails` | `corpusLanguage` | `splitSentences` | `tokenDetails` | `tokenizedDocument` | `topLevelDomains`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Visualize Text Data Using Word Clouds"

Introduced in R2018b

bagOfNgrams

Bag-of-n-grams model

Description

A bag-of-n-grams model records the number of times that each n-gram appears in each document of a collection. An n-gram is a collection of *n* successive words.

`bagOfNgrams` does not split text into words. To create an array of tokenized documents, see `tokenizedDocument`.

Creation

Syntax

```
bag = bagOfNgrams
bag = bagOfNgrams(documents)
bag = bagOfNgrams(___,'NgramLengths',lengths)
bag = bagOfNgrams(uniqueNgrams,counts)
```

Description

`bag = bagOfNgrams` creates an empty bag-of-n-grams model.

`bag = bagOfNgrams(documents)` creates a bag-of-n-grams model and counts the bigrams (pairs of words) in `documents`.

`bag = bagOfNgrams(___,'NgramLengths',lengths)` counts n-grams of the specified lengths using any of the previous syntaxes.

`bag = bagOfNgrams(uniqueNgrams,counts)` creates a bag-of-n-grams model using the n-grams in `uniqueNgrams` and the corresponding frequency counts in `counts`. If `uniqueNgrams` contains `<missing>` values, then the corresponding values in `counts` are ignored.

Input Arguments

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

`uniqueNgrams` — Unique n-gram list
string array | cell array of character vectors

Unique n-gram list, specified as a `NumNgrams`-by-`maxN` string array or cell array of character vectors, where `NumNgrams` is the number of unique n-grams, and `maxN` is the length of the largest n-gram.

The value of `uniqueNgrams(i,j)` is the `j`th word of the `i`th n-gram. If the number of words in the `i`th n-gram is less than `maxN`, then the remaining entries of the `i`th row of `uniqueNgrams` are empty.

If `uniqueNgrams` contains `<missing>`, then the function ignores the corresponding values in `counts`.

Each n-gram must have at least one word.

Example: `["An" ""; "An" "example"; "example" ""]`

Data Types: `string` | `cell`

`counts` — Frequency counts of n-grams
matrix of nonnegative integers

Frequency counts of n-grams corresponding to the rows of `uniqueNgrams`, specified as a matrix of nonnegative integers. The value `counts(i,j)` corresponds to the number of times the n-gram `uniqueNgrams(j,:)` appears in the *i*th document.

`counts` must have as many columns as `uniqueNgrams` has rows.

`lengths` — Lengths of n-grams
2 (default) | positive integer | vector of positive integers

Lengths of n-grams, specified as a positive integer or a vector of positive integers.

Properties

`Counts` — N-gram counts per document
sparse matrix

N-gram counts per document, specified as a sparse matrix.

`Ngrams` — Unique n-grams in model
string array

Unique n-grams in the model, specified as a string array. `Ngrams(i,j)` is the `j`th word of the `i`th n-gram. If the number of columns of `Ngrams` is greater than the number of words in the n-gram, then the remaining entries are empty.

`NgramLengths` — Lengths of n-grams
2 (default) | positive integer | vector of positive integers

Lengths of n-grams, specified as a positive integer or a vector of positive integers.

`Vocabulary` — Unique words in model
string vector

Unique words in the model, specified as a string vector.

Data Types: `string`

`NumNgrams` — Number of n-grams seen
nonnegative integer

Number of n-grams seen, specified as a nonnegative integer.

`NumDocuments` — Number of documents seen
nonnegative integer

Number of documents seen, specified as a nonnegative integer.

Object Functions

encode	Encode documents as matrix of word or n-gram counts
tfidf	Term Frequency–Inverse Document Frequency (tf-idf) matrix
topkngrams	Most frequent n-grams
addDocument	Add documents to bag-of-words or bag-of-n-grams model
removeDocument	Remove documents from bag-of-words or bag-of-n-grams model
removeEmptyDocuments	Remove empty documents from tokenized document array, bag-of-words model, or bag-of-n-grams model
removeNgrams	Remove n-grams from bag-of-n-grams model
removeInfrequentNgrams	Remove infrequently seen n-grams from bag-of-n-grams model
join	Combine multiple bag-of-words or bag-of-n-grams models
wordcloud	Create word cloud chart from text, bag-of-words model, bag-of-n-grams model, or LDA model

Examples

Create Bag-of-N-Grams Model

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
documents(1:10)
```

```
ans = 
  10x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youtl
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair tl
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lov
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt ha
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art bel
```

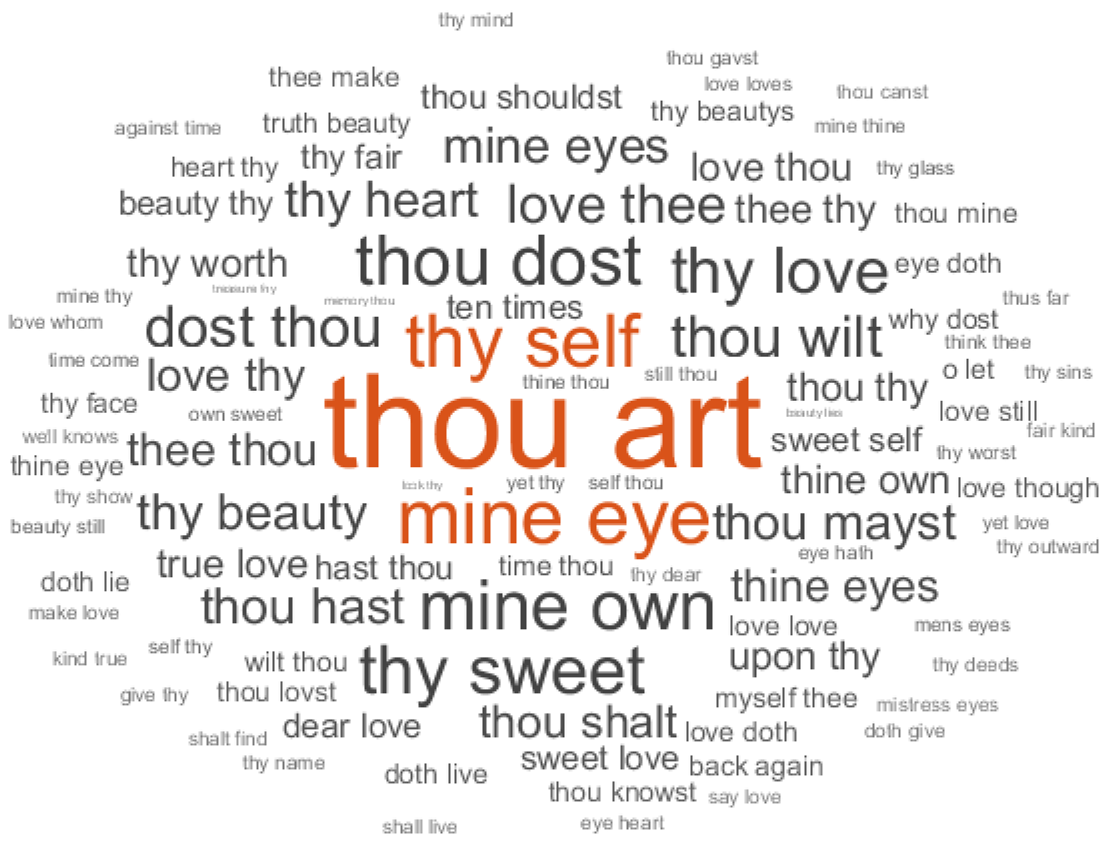
Create a bag-of-n-grams model.

```
bag = bagOfNgrams(documents)
```

```
bag = 
  bagOfNgrams with properties:
         Counts: [154x8799 double]
     Vocabulary: [1x3092 string]
         Ngrams: [8799x2 string]
    NgramLengths: 2
       NumNgrams: 8799
    NumDocuments: 154
```

Visualize the model using a word cloud.

```
figure
wordcloud(bag);
```



Count N-Grams of Different Lengths

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-n-grams model. To count n-grams of length 2 and 3 (bigrams and trigrams), specify `'NgramLengths'` to be the vector `[2 3]`.

```
bag = bagOfNgrams(documents,'NgramLengths',[2 3])
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154×18022 double]
      Vocabulary: [1×3092 string]
          Ngrams: [18022×3 string]
    NgramLengths: [2 3]
       NumNgrams: 18022
    NumDocuments: 154
```

View the 10 most common n-grams of length 2 (bigrams).

```
topkngrams(bag,10,'NGramLengths',2)
```

```
ans=10×3 table
              Ngram                Count    NgramLength
    ___________________________    _____    ___________

    "thou"    "art"        ""       34           2     
    "mine"    "eye"        ""       15           2     
    "thy"     "self"       ""       14           2     
    "thou"    "dost"       ""       13           2     
    "mine"    "own"        ""       13           2     
    "thy"     "sweet"      ""       12           2     
    "thy"     "love"       ""       11           2     
    "dost"    "thou"       ""       10           2     
    "thou"    "wilt"       ""       10           2     
    "love"    "thee"       ""        9           2     
```

View the 10 most common n-grams of length 3 (trigrams).

```
 topkngrams(bag,10,'NGramLengths',3)
```

```
ans=10×3 table
                Ngram                  Count    NgramLength
    _________________________________    _____    ___________

    "thy"     "sweet"    "self"       4           3     
    "why"     "dost"     "thou"       4           3     
    "thy"     "self"     "thy"        3           3     
    "thou"    "thy"      "self"       3           3     
    "mine"    "eye"      "heart"      3           3     
    "thou"    "shalt"    "find"       3           3     
    "fair"    "kind"     "true"       3           3     
    "thou"    "art"      "fair"       2           3     
    "love"    "thy"      "self"       2           3     
    "thy"     "self"     "thou"       2           3     
```

Create Bag-of-N-Grams Model from Unique N-Grams and Counts

Create a bag-of-n-grams model using a string array of unique n-grams and a matrix of counts.

Load the example n-grams and counts from `sonnetsBigramCounts.mat`. This file contains a string array `uniqueNgrams`, which contains the unique n-grams, and the matrix `counts`, which contains the n-gram frequency counts.

```
load sonnetsBigramCounts.mat
```

View the first few n-grams in `uniqueNgrams`.

```
uniqueNgrams(1:10,:)
```

```
ans = 10x2 string
    "fairest"      "creatures"
    "creatures"    "desire"
    "desire"       "increase"
    "increase"     "thereby"
    "thereby"      "beautys"
    "beautys"      "rose"
    "rose"         "might"
    "might"        "never"
    "never"        "die"
    "die"          "riper"
```

Create the bag-of-n-grams model.

```
bag = bagOfNgrams(uniqueNgrams,counts)
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154x8799 double]
      Vocabulary: [1x3092 string]
          Ngrams: [8799x2 string]
    NgramLengths: 2
       NumNgrams: 8799
    NumDocuments: 154
```

See Also

`addDocument` | `bagOfWords` | `encode` | `removeDocument` | `removeEmptyDocuments` | `removeInfrequentNgrams` | `removeNgrams` | `tfidf` | `tokenizedDocument` | `topkngrams`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2018a

bagOfWords

Bag-of-words model

Description

A bag-of-words model (also known as a term-frequency counter) records the number of times that words appear in each document of a collection.

`bagOfWords` does not split text into words. To create an array of tokenized documents, see `tokenizedDocument`.

Creation

Syntax

```
bag = bagOfWords
bag = bagOfWords(documents)
bag = bagOfWords(uniqueWords,counts)
```

Description

`bag = bagOfWords` creates an empty bag-of-words model.

`bag = bagOfWords(documents)` counts the words appearing in `documents` and returns a bag-of-words model.

`bag = bagOfWords(uniqueWords,counts)` creates a bag-of-words model using the words in `uniqueWords` and the corresponding frequency counts in `counts`.

Input Arguments

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

`uniqueWords` — Unique word list
string vector | cell array of character vectors

Unique word list, specified as a string vector or a cell array of character vectors. If `uniqueWords` contains `<missing>`, then the function ignores the missing values. The size of `uniqueWords` must be 1-by-*V* where V is the number of columns of `counts`.

Example: `["an" "example" "list"]`

Data Types: `string` | `cell`

`counts` — Frequency counts of words
matrix of nonnegative integers

Frequency counts of words corresponding to `uniqueWords`, specified as a matrix of nonnegative integers. The value `counts(i,j)` corresponds to the number of times the word `uniqueWords(j)` appears in the *i*th document.

`counts` must have `numel(uniqueWords)` columns.

Properties

`Counts` — Word counts per document
sparse matrix

Word counts per document, specified as a sparse matrix.

`NumDocuments` — Number of documents seen
nonnegative integer

Number of documents seen, specified as a nonnegative integer.

`NumWords` — Number of words in model
nonnegative integer

Number of words in the model, specified as a nonnegative integer.

`Vocabulary` — Unique words in model
string vector

Unique words in the model, specified as a string vector.

Data Types: `string`

Object Functions

encode	Encode documents as matrix of word or n-gram counts
tfidf	Term Frequency–Inverse Document Frequency (tf-idf) matrix
topkwords	Most important words in bag-of-words model or LDA topic
addDocument	Add documents to bag-of-words or bag-of-n-grams model
removeDocument	Remove documents from bag-of-words or bag-of-n-grams model
removeEmptyDocuments	Remove empty documents from tokenized document array, bag-of-words model, or bag-of-n-grams model
removeWords	Remove selected words from documents or bag-of-words model
removeInfrequentWords	Remove words with low counts from bag-of-words model
join	Combine multiple bag-of-words or bag-of-n-grams models
wordcloud	Create word cloud chart from text, bag-of-words model, bag-of-n-grams model, or LDA model

Examples

Create Bag-of-Words Model

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space.

Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

View the top 10 words and their total counts.

```
tbl = topkwords(bag,10)
```

```
tbl=10×2 table
     Word      Count
    _______    _____

    "thy"       281 
    "thou"      234 
    "love"      162 
    "thee"      161 
    "doth"       88 
    "mine"       63 
    "shall"      59 
    "eyes"       56 
    "sweet"      55 
    "time"       53 
```

Create Bag-of-Words Model from Unique Words and Counts

Create a bag-of-words model using a string array of unique words and a matrix of word counts.

```
uniqueWords = ["a" "an" "another" "example" "final" "sentence" "third"];
counts = [ ...
    1 2 0 1 0 1 0;
    0 0 3 1 0 4 0;
    1 0 0 5 0 3 1;
    1 0 0 1 7 0 0];
bag = bagOfWords(uniqueWords,counts)
```

```
bag = 
  bagOfWords with properties:
          Counts: [4x7 double]
      Vocabulary: [1x7 string]
        NumWords: 7
    NumDocuments: 4
```

Import Text from Multiple Files Using a File Datastore

If your text data is contained in multiple files in a folder, then you can import the text data into MATLAB using a file datastore.

Create a file datastore for the example sonnet text files. The examples sonnets have file names "`exampleSonnetN.txt`", where `N` is the number of the sonnet. Specify the read function to be `extractFileText`.

```
readFcn = @extractFileText;
fds = fileDatastore('exampleSonnet*.txt','ReadFcn',readFcn)
```

```
fds = 
  FileDatastore with properties:

                       Files: {
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonne
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonne
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonne
                               ... and 1 more
                              }
                     Folders: {
                              ' ...\Bdoc21a_1606923_5032\ib8F3FCD\26\tp5980089c\textanalytics-ex
                              }
                 UniformRead: 0
                    ReadMode: 'file'
                   BlockSize: Inf
                  PreviewFcn: @extractFileText
      SupportedOutputFormats: [1x16 string]
                     ReadFcn: @extractFileText
    AlternateFileSystemRoots: {}
```

Create an empty bag-of-words model.

```
bag = bagOfWords
```

```
bag = 
  bagOfWords with properties:

          Counts: []
      Vocabulary: [1x0 string]
        NumWords: 0
    NumDocuments: 0
```

Loop over the files in the datastore and read each file. Tokenize the text in each file and add the document to `bag`.

```
while hasdata(fds)
    str = read(fds);
    document = tokenizedDocument(str);
    bag = addDocument(bag,document);
end
```

View the updated bag-of-words model.

```
bag
```

```
bag = 
  bagOfWords with properties:

          Counts: [4x276 double]
      Vocabulary: [1x276 string]
        NumWords: 276
    NumDocuments: 4
```

Remove Stop Words from Bag-of-Words Model

Remove the stop words from a bag-of-words model by inputting a list of stop words to `removeWords`. Stop words are words such as "a", "the", and "in" which are commonly removed from text before analysis.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents);
newBag = removeWords(bag,stopWords)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [2x4 double]
      Vocabulary: ["example"    "short"    "sentence"    "second"]
        NumWords: 4
    NumDocuments: 2
```

Most Frequent Words of Bag-of-Words Model

Create a table of the most frequent words of a bag-of-words model.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Find the top five words.

```
T = topkwords(bag);
```

Find the top 20 words in the model.

```
k = 20;
T = topkwords(bag,k)
```

```
T=20×2 table
      Word      Count
    ________    _____

    "thy"        281 
    "thou"       234 
    "love"       162 
    "thee"       161 
    "doth"        88 
    "mine"        63 
    "shall"       59 
    "eyes"        56 
    "sweet"       55 
    "time"        53 
    "beauty"      52 
    "nor"         52 
    "art"         51 
    "yet"         51 
    "o"           50 
    "heart"       50 
      ⋮
```

Create Tf-idf Matrix

Create a Term Frequency–Inverse Document Frequency (tf-idf) matrix from a bag-of-words model.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
```

```
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Create a tf-idf matrix. View the first 10 rows and columns.

```
M = tfidf(bag);
full(M(1:10,1:10))
```

```
ans = 10×10

    3.6507    4.3438    2.7344    3.6507    4.3438    2.2644    3.2452    3.8918    2.4720    2.
         0         0         0         0         0    4.5287         0         0         0
         0         0         0         0         0         0         0         0         0    2.
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0         0    2.
         0         0    2.7344         0         0         0         0         0         0
```

Create Word Cloud from Bag-of-Words Model

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

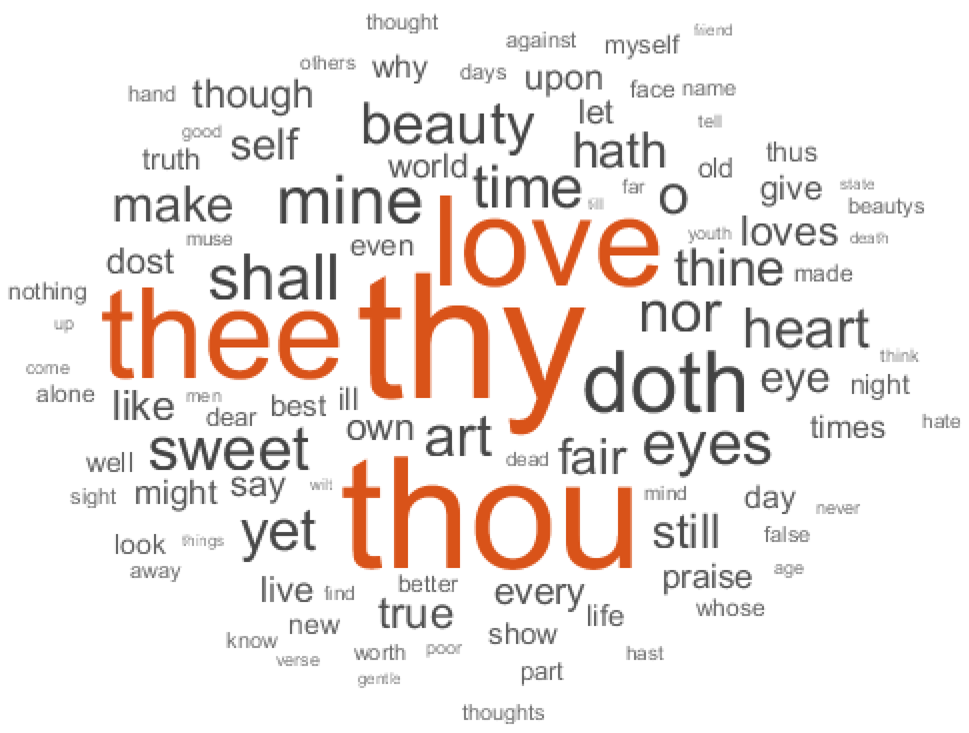
```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
```

```
    NumDocuments: 154
```

Visualize the bag-of-words model using a word cloud.

```
figure
wordcloud(bag);
```



Create Bag-of-Words Model in Parallel

If your text data is contained in multiple files in a folder, then you can import the text data and create a bag-of-words model in parallel using `parfor`. If you have Parallel Computing Toolbox™ installed, then the `parfor` loop runs in parallel, otherwise, it runs in serial. Use `join` to combine an array of bag-of-words models into one model.

Create a bag-of-words model from a collection of files. The examples sonnets have file names "`exampleSonnetN.txt`", where `N` is the number of the sonnet. Get a list of the files and their locations using `dir`.

```
fileLocation = fullfile(matlabroot,'examples','textanalytics','exampleSonnet*.txt');
fileInfo = dir(fileLocation)
```

```
fileInfo = 

  0x1 empty struct array with fields:

    name
    folder
    date
    bytes
    isdir
    datenum
```

Initialize an empty bag-of-words model and then loop over the files and create an array of bag-of-words models.

```
bag = bagOfWords;

numFiles = numel(fileInfo);
parfor i = 1:numFiles
    f = fileInfo(i);
    filename = fullfile(f.folder,f.name);

    textData = extractFileText(filename);
    document = tokenizedDocument(textData);
    bag(i) = bagOfWords(document);
end
```

Combine the bag-of-words models using `join`.

```
bag = join(bag)
```

```
bag = 
  bagOfWords with properties:

          Counts: []
      Vocabulary: [1x0 string]
        NumWords: 0
    NumDocuments: 0
```

Tips

- If you intend to use a held out test set for your work, then partition your text data before using `bagOfWords`. Otherwise, the bag-of-words model may bias your analysis.

See Also

`addDocument` | `bagOfNgrams` | `encode` | `removeDocument` | `removeEmptyDocuments` | `removeInfrequentWords` | `removeWords` | `tfidf` | `tokenizedDocument` | `topkwords`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2017b

bleuEvaluationScore

Evaluate translation or summarization with BLEU similarity score

Syntax

```
score = bleuEvaluationScore(candidate,references)
score = bleuEvaluationScore(candidate,references,'NgramWeights',ngramWeights)
```

Description

The BiLingual Evaluation Understudy (BLEU) scoring algorithm evaluates the similarity between a candidate document and a collection of reference documents. Use the BLEU score to evaluate the quality of document translation and summarization models.

`score = bleuEvaluationScore(candidate,references)` returns the BLEU similarity score between the specified candidate document and the reference documents. The function computes n-gram overlaps between `candidate` and `references` for n-gram lengths one through four, with equal weighting. For more information, see “BLEU Score” on page 1-52.

`score = bleuEvaluationScore(candidate,references,'NgramWeights',ngramWeights)` uses the specified n-gram weighting, where `ngramWeights(i)` corresponds to the weight for n-grams of length `i`. The length of the weight vector determines the range of n-gram lengths to use for the BLEU score evaluation.

Examples

Evaluate Summary

Create an array of tokenized documents and extract a summary using the `extractSummary` function.

```
str = [
    "The fox jumped over the dog."
    "The fast brown fox jumped over the lazy dog."
    "The lazy dog saw a fox jumping."
    "There seem to be animals jumping other animals."
    "There are quick animals and lazy animals"];
documents = tokenizedDocument(str);
summary = extractSummary(documents)
```

```
summary = 
  tokenizedDocument:

   10 tokens: The fast brown fox jumped over the lazy dog .
```

Specify the reference documents as a `tokenizedDocument` array.

```
str = [
    "The quick brown animal jumped over the lazy dog."
    "The quick brown fox jumped over the lazy dog."];
references = tokenizedDocument(str);
```

Calculate the BLEU score between the summary and the reference documents using the `bleuEvaluationScore` function.

```
score = bleuEvaluationScore(summary,references)
```

```
score = 0.7825
```

This score indicates a fairly good similarity. A BLEU score close to one indicates strong similarity.

Specify N-Gram Weights

Create an array of tokenized documents and extract a summary using the `extractSummary` function.

```
str = [
    "The fox jumped over the dog."
    "The fast brown fox jumped over the lazy dog."
    "The lazy dog saw a fox jumping."
    "There seem to be animals jumping other animals."
    "There are quick animals and lazy animals"];
documents = tokenizedDocument(str);
summary = extractSummary(documents)
```

```
summary = 
  tokenizedDocument:

   10 tokens: The fast brown fox jumped over the lazy dog .
```

Specify the reference documents as a `tokenizedDocument` array.

```
str = [
    "The quick brown animal jumped over the lazy dog."
    "The quick brown fox jumped over the lazy dog."];
references = tokenizedDocument(str);
```

Calculate the BLEU score between the candidate document and the reference documents using the default options. The `bleuEvaluationScore` function, by default, uses n-grams of length one through four with equal weights.

```
score = bleuEvaluationScore(summary,references)
```

```
score = 0.7825
```

Given that the summary document differs only by one word to one of the reference documents, this score might suggest a lower similarity than might be expected. This behavior is due to the function using n-grams which are too large for the short document length.

To address this, use shorter n-grams by setting the `'NgramWeights'` option to a shorter vector. Calculate the BLEU score again using only unigrams and bigrams by setting the `'NgramWeights'` option to a two-element vector. Treat unigrams and bigrams equally by specifying equal weights.

```
score = bleuEvaluationScore(summary,references,'NgramWeights',[0.5 0.5])
```

```
score = 0.8367
```

This score suggests a better similarity than before.

Input Arguments

`candidate` — Candidate document
`tokenizedDocument` scalar | string array | cell array of character vectors

Candidate document, specified as a `tokenizedDocument` scalar, a string array, or a cell array of character vectors. If `candidate` is not a `tokenizedDocument` scalar, then it must be a row vector representing a single document, where each element is a word.

`references` — Reference documents
`tokenizedDocument` array | string array | cell array of character vectors

Reference documents, specified as a `tokenizedDocument` array, a string array, or a cell array of character vectors. If `references` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To evaluate against multiple reference documents, use a `tokenizedDocument` array.

`ngramWeights` — N-gram weights
`[0.25 0.25 0.25 0.25]` (default) | row vector of finite nonnegative values

N-gram weights, specified as a row vector of finite nonnegative values, where `ngramWeights(i)` corresponds to the weight for n-grams of length `i`. The length of the weight vector determines the range of n-gram lengths to use for the BLEU score evaluation. The function normalizes the n-gram weights to sum to one.

Tip If the number of words in `candidate` is smaller than the number of elements in `ngramWeights`, then the resulting BLEU score is zero. To ensure that `bleuEvaluationScore` returns nonzero scores for very short documents, set `ngramWeights` to a vector with fewer elements than the number of words in `candidate`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — BLEU score
scalar

BLEU score, returned as a scalar value in the range [0,1] or `NaN`.

A BLEU score close to zero indicates poor similarity between `candidate` and `references`. A BLEU score close to one indicates strong similarity. If `candidate` is identical to one of the reference documents, then `score` is 1. If `candidate` and `references` are both empty documents, then `score` is `NaN`. For more information, see “BLEU Score” on page 1-52.

Tip If the number of words in `candidate` is smaller than the number of elements in `ngramWeights`, then the resulting BLEU score is zero. To ensure that `bleuEvaluationScore` returns nonzero scores for very short documents, set `ngramWeights` to a vector with fewer elements than the number of words in `candidate`.

Algorithms

BLEU Score

The BiLingual Evaluation Understudy (BLEU) scoring algorithm [1] evaluates the similarity between a candidate document and a collection of reference documents. Use the BLEU score to evaluate the quality of document translation and summarization models.

To compute the BLEU score, the algorithm uses n-gram counts, *clipped n-gram counts*, *modified n-gram precision scores*, and a *brevity penalty*.

The clipped n-gram counts function $\text{Count}_{\text{clip}}$, if necessary, truncates the n-gram count for each n-gram so that it does not exceed the largest count observed in any single reference for that n-gram. The clipped counts function is given by

$$\text{Count}_{\text{clip}}(\text{n-gram}) = \min(\text{Count}(\text{n-gram}), \text{MaxRefCount}(\text{n-gram})),$$

where Count(n-gram) denotes the n-gram counts and MaxRefCount(n-gram) is the largest n-gram count observed in a single reference document for that n-gram.

The *modified n-gram precision scores* are given by

$$p_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{\text{n-gram} \in C} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{C' \in \{\text{Candidates}\}} \sum_{\text{n-gram}' \in C'} \text{Count}(\text{n-gram}')},$$

where n corresponds to the n-gram length and {candidates} is the set of sentences in the candidate documents.

Given a vector of n-gram weights w, the *BLEU score* is given by

$$\text{bleuScore} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log \bar{p}_n\right),$$

where N is the largest n-gram length, the entries in $\bar{p}$ correspond to the geometric averages of the modified n-gram precisions, and BP is the *brevity penalty* given by

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1 - \frac{r}{c}} & \text{if } c \le r \end{cases}$$

where c is the length of the candidate document and r is the length of the reference document with length closest to the candidate length.

References

[1] Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "BLEU: A Method for Automatic Evaluation of Machine Translation." In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311-318. Association for Computational Linguistics, 2002.

See Also

`bm25Similarity` | `cosineSimilarity` | `extractSummary` | `lexrankScores` | `mmrScores` | `rougeEvaluationScore` | `textrankScores` | `tokenizedDocument`

Topics
"Sequence-to-Sequence Translation Using Attention"

Introduced in R2020a

bm25Similarity

Document similarities with BM25 algorithm

Syntax

```
similarities = bm25Similarity(documents)
similarities = bm25Similarity(documents,queries)

similarities = bm25Similarity(bag)
similarities = bm25Similarity(bag,queries)

similarities = bm25Similarity( ___ ,Name,Value)
```

Description

Use `bm25Similarity` to calculate document similarities.

By default, this function calculates BM25 similarities. To calculate BM11, BM15, or BM25+ similarities, use the `'DocumentLengthScaling'` and `'DocumentLengthCorrection'` arguments.

`similarities = bm25Similarity(documents)` returns the pairwise BM25 similarities between the specified documents. The score in `similarities(i,j)` represents the similarity between `documents(i)` and `documents(j)`.

`similarities = bm25Similarity(documents,queries)` returns similarities between `documents` and `queries`. The score in `similarities(i,j)` represents the similarity between `documents(i)` and `queries(j)`.

`similarities = bm25Similarity(bag)` returns similarities between the documents encoded by the specified bag-of-words or bag-of-n-grams model. The score in `similarities(i,j)` represents the similarity between the `i`th and `j`th documents encoded by `bag`.

`similarities = bm25Similarity(bag,queries)` returns similarities between the documents encoded by the bag-of-words or bag-of-n-grams model `bag` and the documents specified by `queries`. The score in `similarities(i,j)` represents the similarity between the `i`th document encoded by `bag` and `queries(j)`.

`similarities = bm25Similarity( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments. For instance, to use the BM25+ algorithm, set the `'DocumentLengthCorrection'` option to a nonzero value.

Examples

Similarity Between Documents

Create an array of tokenized documents.

```
textData = [
    "the quick brown fox jumped over the lazy dog"
    "the fast brown fox jumped over the lazy dog"
    "the lazy dog sat there and did nothing"
    "the other animals sat there watching"];
documents = tokenizedDocument(textData)
```

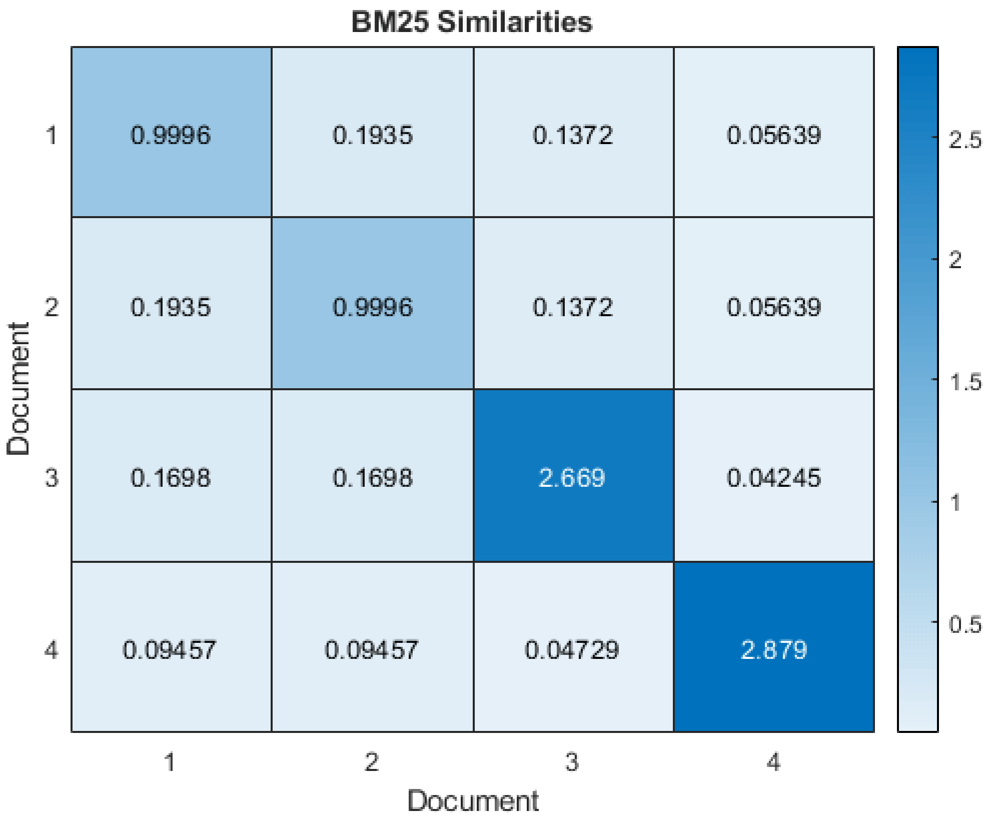
```
documents = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    9 tokens: the fast brown fox jumped over the lazy dog
    8 tokens: the lazy dog sat there and did nothing
    6 tokens: the other animals sat there watching
```

Calculate the similarities between them using the `bm25Similarity` function. The output is a sparse matrix.

```
similarities = bm25Similarity(documents);
```

Visualize the similarities of the documents in a heat map.

```
figure
heatmap(similarities);
xlabel("Document")
ylabel("Document")
title("BM25 Similarities")
```



The first three documents have the highest pairwise similarities which indicates that these documents are most similar. The last document has comparatively low pairwise similarities with the other documents which indicates that this document is less like the other documents.

Similarity to Query

Create an array of input documents.

```
str = [
    "the quick brown fox jumped over the lazy dog"
    "the fast fox jumped over the lazy dog"
    "the dog sat there and did nothing"
    "the other animals sat there watching"];
documents = tokenizedDocument(str)
```

```
documents = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    8 tokens: the fast fox jumped over the lazy dog
    7 tokens: the dog sat there and did nothing
    6 tokens: the other animals sat there watching
```

Create an array of query documents.

```
str = [
    "a brown fox leaped over the lazy dog"
    "another fox leaped over the dog"];

queries = tokenizedDocument(str)
```

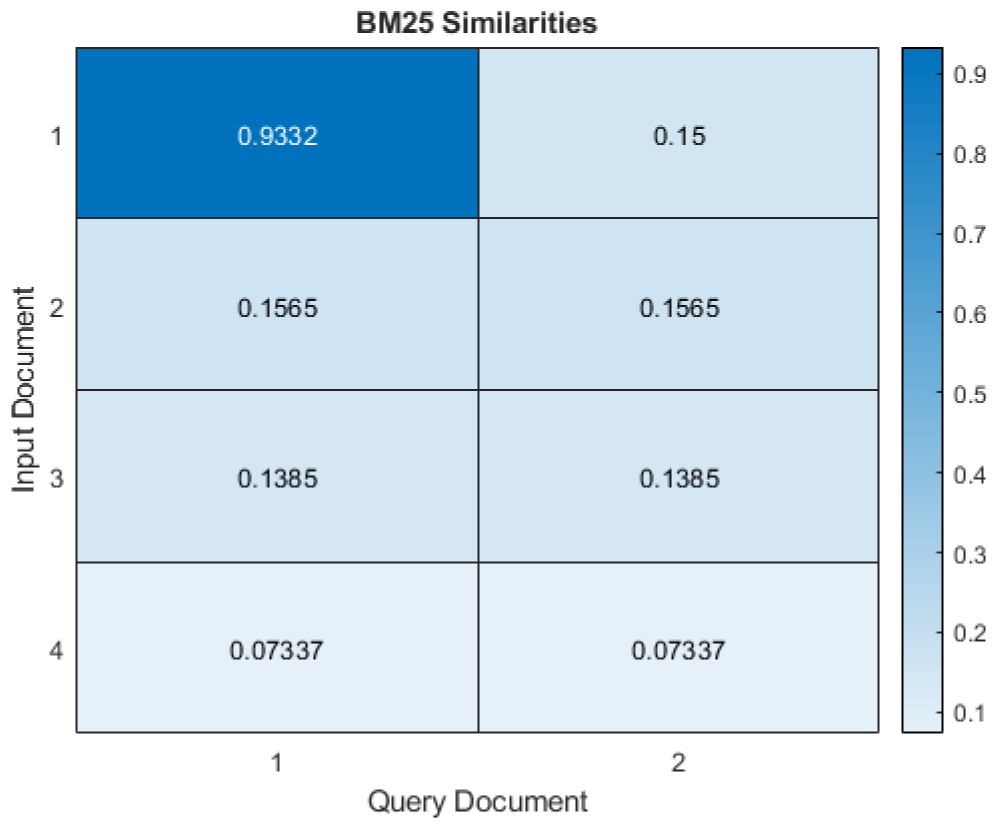
```
queries = 
  2x1 tokenizedDocument:

    8 tokens: a brown fox leaped over the lazy dog
    6 tokens: another fox leaped over the dog
```

Calculate the similarities between input documents and query documents using the `bm25Similarity` function. The output is a sparse matrix. The score in `similarities(i,j)` represents the similarity between `documents(i)` and `queries(j)`.

```
similarities = bm25Similarity(documents,queries);
```

Visualize the similarities of the documents in a heat map.

```
figure
heatmap(similarities);
xlabel("Query Document")
ylabel("Input Document")
title("BM25 Similarities")
```



In this case, the first input document is most like the first query document.

Document Similarities Using Bag-of-Words Model

Create a bag-of-words model from the text data in `sonnets.csv`.

```
filename = "sonnets.csv";
tbl = readtable(filename,'TextType','string');
textData = tbl.Sonnet;
documents = tokenizedDocument(textData);
bag = bagOfWords(documents)
```

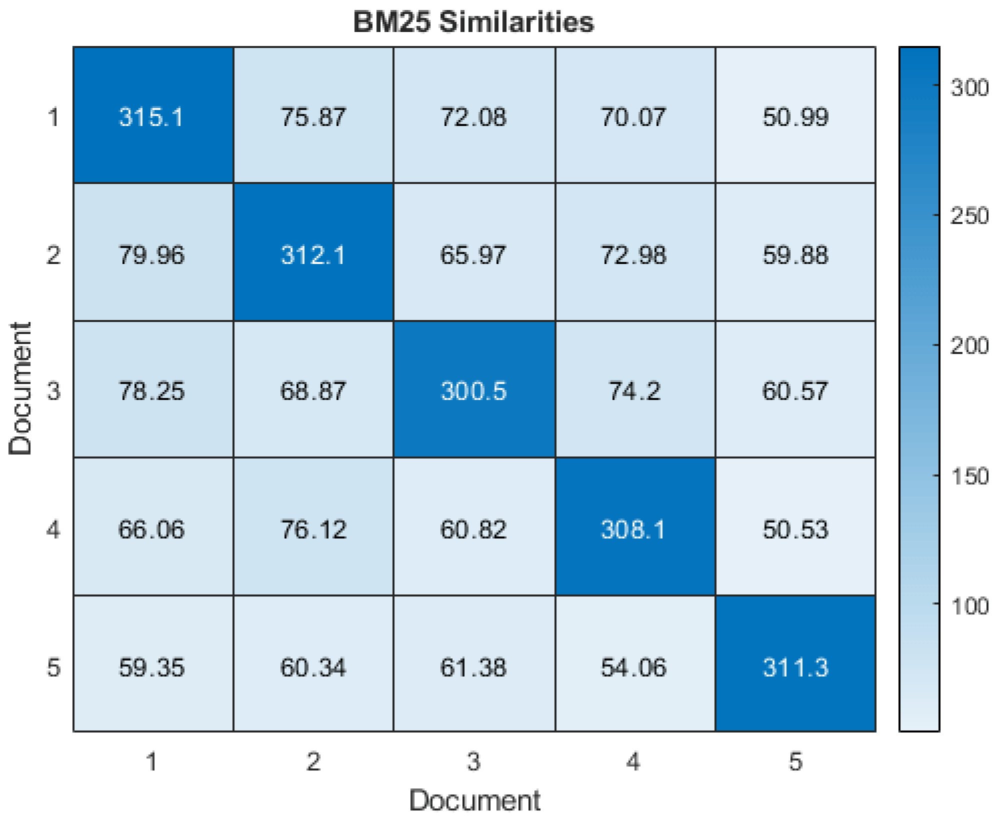
```
bag = 
  bagOfWords with properties:

          Counts: [154x3527 double]
      Vocabulary: [1x3527 string]
        NumWords: 3527
    NumDocuments: 154
```

Calculate similarities between the sonnets using the `bm25Similarity` function. The output is a sparse matrix.

```
similarities = bm25Similarity(bag);
```

Visualize the similarities between the first five documents in a heat map.

```
figure
heatmap(similarities(1:5,1:5));
xlabel("Document")
ylabel("Document")
title("BM25 Similarities")
```



Evaluate BM25+ Document Similarity

The BM25+ algorithm addresses a limitation of the BM25 algorithm: the component of the term-frequency normalization by document length is not properly lower bounded. As a result of this limitation, long documents which do not match the query term can often be scored unfairly by BM25 as having a similar relevance to shorter documents that do not contain the query term.

BM25+ addresses this limitation by using a document length correction factor (the value of the `'DocumentLengthScaling'` name-value pair). This factor prevents the algorithm from over-penalizing long documents.

Create two arrays of tokenized documents.

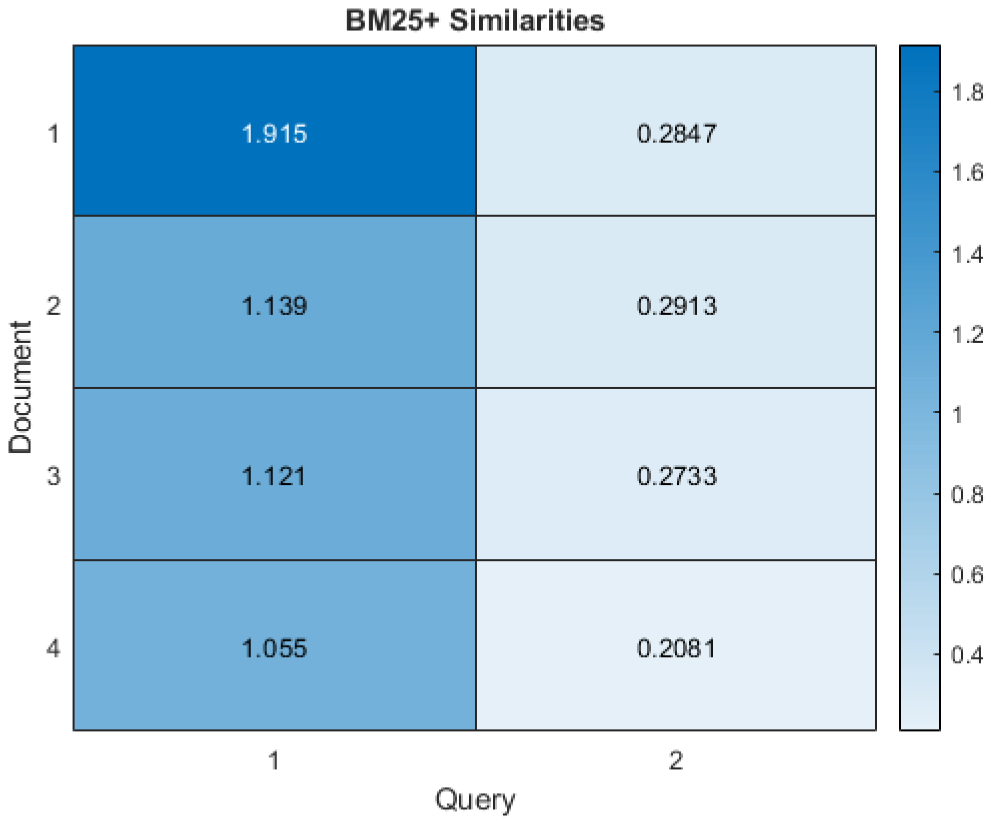
```
textData1 = [
    "the quick brown fox jumped over the lazy dog"
    "the fast fox jumped over the lazy dog"
    "the dog sat there and did nothing"
    "the other animals sat there watching"];
documents1 = tokenizedDocument(textData1)

documents1 = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    8 tokens: the fast fox jumped over the lazy dog
    7 tokens: the dog sat there and did nothing
    6 tokens: the other animals sat there watching


textData2 = [
    "a brown fox leaped over the lazy dog"
    "another fox leaped over the dog"];
documents2 = tokenizedDocument(textData2)

documents2 = 
  2x1 tokenizedDocument:

    8 tokens: a brown fox leaped over the lazy dog
    6 tokens: another fox leaped over the dog
```

To calculate the BM25+ document similarities, use the `bm25Similarity` function and set the `'DocumentLengthCorrection'` option to a nonzero value. In this case, set the `'DocumentLengthCorrection'` option to 1.

```
similarities = bm25Similarity(documents1,documents2,'DocumentLengthCorrection',1);
```

Visualize the similarities of the documents in a heat map.

```
figure
heatmap(similarities);
xlabel("Query")
ylabel("Document")
title("BM25+ Similarities")
```



Here, when compared with the example “Similarity Between Documents” on page 1-54, the scores show more similarity between the input documents and the first query document.

Input Arguments

`documents` — Input documents

`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

`bag` — Input model

`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`queries` — Set of query documents

`tokenizedDocument` array | `bagOfWords` object | `bagOfNgrams` object | string array of words | cell array of character vectors

Set of query documents, specified as one of the following:

- A `tokenizedDocument` array
- A `bagOfWords` or `bagOfNgrams` object
- A 1-by-*N* string array representing a single document, where each element is a word
- A 1-by-*N* cell array of character vectors representing a single document, where each element is a word

To compute term frequency and inverse document frequency statistics, the function encodes `queries` using a bag-of-words model. The model it uses depends on the syntax you call it with. If your syntax specifies the input argument `documents`, then it uses `bagOfWords(documents)`. If your syntax specifies `bag`, then it uses `bag`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `bm25Similarity(documents,'TFScaling',1.5)` returns the pairwise similarities for the specified documents and sets the token frequency scaling factor to 1.5.

`IDFWeight` — Method to compute inverse document frequency factor
`'textrank'` (default) | `'classic-bm25'` | `'normal'` | `'unary'` | `'smooth'` | `'max'` | `'probabilistic'`

Method to compute inverse document frequency factor, specified as the comma-separated pair consisting of `'IDFWeight'` and one of the following:

- `'textrank'` – Use TextRank IDF weighting [2]. For each term, set the IDF factor to
  - `log((N-NT+0.5)/(NT+0.5))` if the term occurs in more than half of the documents, where `N` is the number of documents in the input data and `NT` is the number of documents in the input data containing each term.
  - `IDFCorrection*avgIDF` if the term occurs in half of the documents or f, where `avgIDF` is the average IDF of all tokens.
- `'classic-bm25'` – For each term, set the IDF factor to `log((N-NT+0.5)/(NT+0.5))`.
- `'normal'` – For each term, set the IDF factor to `log(N/NT)`.
- `'unary'` – For each term, set the IDF factor to 1.
- `'smooth'` – For each term, set the IDF factor to `log(1+N/NT)`.
- `'max'` – For each term, set the IDF factor to `log(1+max(NT)/NT)`.
- `'probabilistic'` – For each term, set the IDF factor to `log((N-NT)/NT)`.

where `N` is the number of documents in the input data and `NT` is the number of documents in the input data containing each term.

`TFScaling` — Term frequency scaling factor
1.2 (default) | nonnegative scalar

Term frequency scaling factor, specified as the comma-separated pair consisting of `'TFScaling'` and a nonnegative scalar.

This option corresponds to the value *k* in the BM25 algorithm. For more information, see “BM25” on page 1-63.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`DocumentLengthScaling` — Document length scaling factor
0.75 (default) | scalar in the range [0,1]

Document length scaling factor, specified as the comma-separated pair consisting of `'DocumentLengthScaling'` and a scalar in the range [0,1].

This option corresponds to the value *b* in the BM25 algorithm. When *b=1*, the BM25 algorithm is equivalent to BM11. When *b=0*, the BM25 algorithm is equivalent to BM15. For more information, see “BM11” on page 1-64, “BM15” on page 1-64, or “BM25” on page 1-63.

Data Types: `double`

`IDFCorrection` — Inverse document frequency correction factor
0.25 (default) | nonnegative scalar

Inverse document frequency correction factor, specified as the comma-separated pair consisting of `'IDFCorrection'` and a nonnegative scalar.

This option only applies when `'IDFWeight'` is `'textrank'`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`DocumentLengthCorrection` — Document length correction factor
0 (default) | nonnegative scalar

Document length correction factor, specified as the comma-separated pair consisting of `'DocumentLengthCorrection'` and a nonnegative scalar.

This option corresponds to the value *δ* in the BM25+ algorithm. If the document length correction factor is nonzero, then the `bm25Similarity` function uses the BM25+ algorithm. Otherwise, the function uses the BM25 algorithm. For more information, see “BM25+” on page 1-63.

Data Types: `double`

Output Arguments

`similarities` — BM25 similarity scores
sparse matrix

BM25 similarity scores, returned as a sparse matrix:

- Given a single array of tokenized documents, `similarities` is a *N*-by-*N* nonsymmetric matrix, where `similarities(i,j)` represents the similarity between `documents(i)` and `documents(j)`, and N is the number of input documents.
- Given an array of tokenized documents and a set of query documents, `similarities` is an *N1*-by-*N2* matrix, where `similarities(i,j)` represents the similarity between `documents(i)` and the `j`th query document, and *N1* and *N2* represents the number of documents in `documents` and `queries`, respectively.
- Given a single bag-of-words or bag-of-n-grams model, `similarities` is a `bag.NumDocuments`-by-`bag.NumDocuments` nonsymmetric matrix, where `similarities(i,j)` represents the similarity between the `i`th and `j`th documents encoded by `bag`.
- Given a bag-of-words or bag-of-n-grams models and a set of query documents, `similarities` is a `bag.NumDocuments`-by-*N2* matrix, where `similarities(i,j)` represents the similarity

between the `i`th document encoded by `bag` and the `j`th document in `queries`, and *N2* corresponds to the number of documents in `queries`.

Tips

- The BM25 algorithm aggregates and uses information from all the documents in the input data via the term frequency (TF) and inverse document frequency (IDF) based options. This behavior means that the same pair of documents can yield different BM25 similarity scores when the function is given different collections of documents.
- The BM25 algorithm can output different scores when comparing documents to themselves. This behavior is due to the use of the IDF weights and the document length in the BM25 algorithm.

Algorithms

BM25

Given a document from a collection of documents D, and a query document, the BM25 score is given by

$$\mathrm{BM25}(\mathrm{document}, \mathrm{query}; \mathrm{D}) = \sum_{\mathrm{word} \in \mathrm{query}} \left(\mathrm{IDF}(\mathrm{word}; \mathrm{D}) \frac{\mathrm{Count}(\mathrm{word}, \mathrm{document})(k+1)}{\mathrm{Count}(\mathrm{word}, \mathrm{document}) + k\left(1 - b + b\frac{|\mathrm{document}|}{\bar{n}}\right)} \right),$$

where

- *Count(word,document)* denotes the frequency of *word* in *document*.
- $\bar{n}$ denotes the average document length in D.
- k denotes the *term frequency scaling factor* (the value of the `'TFScaling'` name-value pair argument). This factor dampens the influence of frequently appearing terms on the BM25 score.
- b denotes the *document length scaling factor* (the value of the `'DocumentLengthScaling'` name-value pair argument). This factor controls how the length of a document influences the BM25 score. When *b=1*, the BM25 algorithm is equivalent to BM11. When *b=0*, the BM25 algorithm is equivalent to BM15.
- $\mathrm{IDF}(\mathrm{word}, \mathrm{D})$ is the inverse document frequency of the specified word given the collection of documents D.

BM25+

The BM25+ algorithm addresses a limitation of the BM25 algorithm: the component of the term-frequency normalization by document length is not properly lower bounded. As a result of this limitation, long documents which do not match the query term can often be scored unfairly by BM25 as having a similar relevance to shorter documents that do not contain the query term.

The BM25+ algorithm is the same as the BM25 algorithm with one extra parameter. Given a document from a collection of documents D and a query document, the BM25+ score is given by

$$\mathrm{BM25}^{+}(\mathrm{document}, \mathrm{query}; \mathrm{D}) = \sum_{\mathrm{word} \in \mathrm{query}} \left(\mathrm{IDF}(\mathrm{word};\mathrm{D}) \left(\frac{\mathrm{Count}(\mathrm{word}, \mathrm{document})(k+1)}{\mathrm{Count}(\mathrm{word}, \mathrm{document}) + k\left(1 - b + b\frac{|\mathrm{document}|}{\bar{n}}\right)} + \delta \right) \right),$$

where the extra parameter δ denotes the *document length correction factor* (the value of the `'DocumentLengthScaling'` name-value pair). This factor prevents the algorithm from over-penalizing long documents.

BM11

BM11 is a special case of “BM25” on page 1-63 when *b=1*.

Given a document from a collection of documents D, and a query document, the BM11 score is given by

$$\mathrm{BM11}(\mathrm{document}, \mathrm{query}; \mathrm{D}) = \sum_{\mathrm{word} \in \mathrm{query}} \left(\mathrm{IDF}(\mathrm{word};\mathrm{D}) \frac{\mathrm{Count}(\mathrm{word}, \mathrm{document})(k+1)}{\mathrm{Count}(\mathrm{word}, \mathrm{document}) + k\left(\frac{|\mathrm{document}|}{\bar{n}}\right)} \right).$$

BM15

BM15 is a special case of “BM25” on page 1-63 when *b=0*.

Given a document from a collection of documents D, and a query document, the BM15 score is given by

$$\mathrm{BM15}(\mathrm{document}, \mathrm{query}; \mathrm{D}) = \sum_{\mathrm{word} \in \mathrm{query}} \left(\mathrm{IDF}(\mathrm{word};\mathrm{D}) \frac{\mathrm{Count}(\mathrm{word}, \mathrm{document})(k+1)}{\mathrm{Count}(\mathrm{word}, \mathrm{document}) + k} \right).$$

References

[1] Robertson, Stephen, and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond." *Foundations and Trends® in Information Retrieval* 3, no. 4 (2009): 333-389.

[2] Barrios, Federico, Federico López, Luis Argerich, and Rosa Wachenchauzer. "Variations of the Similarity Function of TextRank for Automated Summarization." *arXiv preprint arXiv:1602.03606* (2016).

See Also

`bleuEvaluationScore` | `cosineSimilarity` | `extractSummary` | `lexrankScores` | `mmrScores` | `rougeEvaluationScore` | `textrankScores` | `tokenizedDocument`

Topics

“Sequence-to-Sequence Translation Using Attention”

Introduced in R2020a

characterCategories

Package: `textanalytics.unicode`

Unicode character categories

Syntax

```
ucats = characterCategories(str32)
ucats = characterCategories(str32,'Granularity',granularity)
```

Description

`ucats = characterCategories(str32)` returns the major Unicode character categories for the characters in the `UTF32` object `str`.

`ucats = characterCategories(str32,'Granularity',granularity)` also specifies the granularity of the returned categories. For example, `characterCategories(str32,'Granularity','detailed')` returns detailed Unicode character categories.

Examples

Get Unicode Character Categories

Convert the string `"Hello! □□"` to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! □□";
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Get the Unicode character categories of `str32` using the `characterCategories` function.

```
ucats = characterCategories(str32)
```

```
ucats = 1x1 cell array
    {[L    L    L    L    L    P    Z    S]}
```

The Unicode character categories "L", "P", "Z", and "S" correspond to "letter", "punctuation", "separator", and "symbol", respectively.

Get Detailed Unicode Character Categories

Convert the string `"Hello! 😀"` to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! 😀";
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Get the Unicode character categories of `str32` using the `characterCategories` function. To return detailed Unicode character categories, set the `'Granularity'` option to `'detailed'`.

```
ucats = characterCategories(str32,'Granularity','detailed')
```

```
ucats = 1x1 cell array
    {[Lu    Ll    Ll    Ll    Ll    Po    Zs    So]}
```

The Unicode character categories "Lu", "Ll", "Po", "Zs", and "So" correspond to "uppercase letter", "lowercase letter", "other punctuation", "space separator", and "other symbol", respectively.

Input Arguments

`str32` — UTF-32 string representation
UTF32 array

UTF-32 string representation, specified as a `UTF32` array.

`granularity` — Granularity of returned Unicode character categories
`'major'` (default) | `'detailed'`

Granularity of returned Unicode character categories, specified as one of the following:

- `'major'` – Return the major Unicode character category. This includes the first character of the Unicode character category only.
- `'detailed'` – Return detailed Unicode character codes. This includes all characters of the Unicode character category.

Output Arguments

`ucats` — Unicode character categories
cell array of categorical vectors

Unicode character categories, returned as a cell array of categorical vectors.

This table shows the major and detailed Unicode character categories. To specify which granularity of Unicode character categories to return, use the `Granularity` option.

Major Character Category	Major Character Category Description	Detailed Character Category	Detailed Character Category Description
L	Letter	Lu	Uppercase letter
		Ll	Lowercase letter
		Lt	Titlecase letter
		Lm	Modifier letter
		Lo	Other letter
M	Mark	Mn	Nonspacing mark
		Mc	Spacing mark
		Me	Enclosing mark
N	Number	Nd	Decimal number
		Nl	Letter number
		No	Other number
P	Punctuation	Pc	Connector punctuation
		Pd	Dash punctuation
		Ps	Open punctuation
		Pe	Close punctuation
		Pi	Initial punctuation
		Pf	Final punctuation
		Po	Other punctuation
S	Symbol	Sm	Math symbol
		Sc	Currency symbol
		Sk	Modifier symbol
		So	Other symbol
Z	Separator	Zs	Space separator
		Zl	Line separator
		Zp	Paragraph separator
C	Other	Cc	Control
		Cf	Format
		Cs	Surrogate
		Co	Private use
		Cn	Unassigned

References

[1] *Unicode® Standard Annex #44 Unicode Character Database* https://www.unicode.org/reports/tr44

See Also

`characterCategories` | `hex` | `textanalytics.unicode.nfd` | `tokenizedDocument`

Topics
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Language Considerations"

Introduced in R2021a

context

Search documents for word or n-gram occurrences in context

Syntax

```
T = context(documents,word)
T = context(documents,ngram)
T = context( ___ ,contextLength)
T = context( ___ ,Name,Value)
```

Description

`T = context(documents,word)` searches for occurrences of a single word in `documents` and returns a table showing `word` in context and its locations. The function, by default, is case sensitive.

`T = context(documents,ngram)` searches for occurrences of an n-gram in `documents`. The function, by default, is case sensitive.

`T = context( ___ ,contextLength)` specifies the length of the context to return using any of the previous syntaxes.

`T = context( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments using any of the previous syntaxes.

Examples

Search Documents for Word Occurrences

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Search for the word "life".

```
tbl = context(documents,"life");
head(tbl)
```

```
ans=8×3 table
                             Context                             Document    Word
    _________________________________________________________    ________    ____

    "consumst thy self single life ah thou issueless shalt "         9        10 
    "ainted counterfeit lines life life repair times pencil"        16        35 
    "d counterfeit lines life life repair times pencil pupi"        16        36 
    " heaven knows tomb hides life shows half parts write b"       17        14
    "he eyes long lives gives life thee                    "       18        69
    "tender embassy love thee life made four two alone sink"       45        23
    "ves beauty though lovers life beauty shall black lines"       63        50
    "s shorn away live second life second head ere beautys "       68        27
```

View the occurrences in a string array.

```
tbl.Context
```

```
ans = 23x1 string
    "consumst thy self single life ah thou issueless shalt "
    "ainted counterfeit lines life life repair times pencil"
    "d counterfeit lines life life repair times pencil pupi"
    " heaven knows tomb hides life shows half parts write b"
    "he eyes long lives gives life thee                    "
    "tender embassy love thee life made four two alone sink"
    "ves beauty though lovers life beauty shall black lines"
    "s shorn away live second life second head ere beautys "
    "e rehearse let love even life decay lest wise world lo"
    "st bail shall carry away life hath line interest memor"
    "art thou hast lost dregs life prey worms body dead cow"
    "           thoughts food life sweetseasond showers gro"
    "tten name hence immortal life shall though once gone w"
    " beauty mute others give life bring tomb lives life fa"
    "ve life bring tomb lives life fair eyes poets praise d"
    " steal thyself away term life thou art assured mine li"
    "fe thou art assured mine life longer thy love stay dep"
    " fear worst wrongs least life hath end better state be"
    "anst vex inconstant mind life thy revolt doth lie o ha"
    " fame faster time wastes life thou preventst scythe cr"
    "ess harmful deeds better life provide public means pub"
    "ate hate away threw savd life saying                  "
    " many nymphs vowd chaste life keep came tripping maide"
```

Search Documents for N-Gram Occurrences

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Search for the bigram "dost thou".

```
ngram = ["dost" "thou"];
tbl = context(documents,ngram);
head(tbl)
```

```
ans=8×3 table
                              Context                                Document      Word
```

```
    _____________________________________________________________    ________    ________

    "unthrifty loveliness why dost thou spend upon thy self thy "        4           4     5
    "ee beauteous niggard why dost thou abuse bounteous largess "        4          25    26
    "ve profitless usurer why dost thou great sum sums yet canst"        4          35    36
    "eavy eyelids weary night dost thou desire slumbers broken s"       61          10    11
    "             sweet lovely dost thou make shame like canker f"       95           3     4
    "hy budding name o sweets dost thou thy sins enclose tongue "       95          19    20
    "ruth beauty love depends dost thou therein dignified make a"      101          16    17
    "    thou blind fool love dost thou mine eyes behold know be"      137           5     6
```

View the occurrences in a string array.

```
tbl.Context
```

```
ans = 10x1 string
    "unthrifty loveliness why dost thou spend upon thy self thy "
    "ee beauteous niggard why dost thou abuse bounteous largess "
    "ve profitless usurer why dost thou great sum sums yet canst"
    "eavy eyelids weary night dost thou desire slumbers broken s"
    "             sweet lovely dost thou make shame like canker f"
    "hy budding name o sweets dost thou thy sins enclose tongue "
    "ruth beauty love depends dost thou therein dignified make a"
    "    thou blind fool love dost thou mine eyes behold know be"
    "h rebel powers array why dost thou pine suffer dearth paint"
    "y large cost short lease dost thou upon thy fading mansion "
```

Specify Context Length

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Search for the word "life" and return each occurrence with a 15-character context before and after.

```
tbl = context(documents,"life",15);
head(tbl)
```

```
ans=8×3 table
                  Context                    Document    Word
    ____________________________________    ________    ____

    "hy self single life ah thou issuel"        9        10
    "nterfeit lines life life repair ti"       16        35
    "eit lines life life repair times p"       16        36
    "ows tomb hides life shows half par"       17        14
    "ng lives gives life thee          "       18        69
    "assy love thee life made four two "       45        23
    " though lovers life beauty shall b"       63        50
    "ay live second life second head er"       68        27
```

View the occurrences in a string array.

```
tbl.Context
```

```
ans = 23x1 string
    "hy self single life ah thou issuel"
    "nterfeit lines life life repair ti"
    "eit lines life life repair times p"
    "ows tomb hides life shows half par"
    "ng lives gives life thee          "
    "assy love thee life made four two "
    " though lovers life beauty shall b"
    "ay live second life second head er"
    " let love even life decay lest wis"
    "all carry away life hath line inte"
    "ast lost dregs life prey worms bod"
    " thoughts food life sweetseasond s"
    "hence immortal life shall though o"
    "te others give life bring tomb liv"
    "ing tomb lives life fair eyes poet"
    "self away term life thou art assur"
    "t assured mine life longer thy lov"
    "t wrongs least life hath end bette"
    "nconstant mind life thy revolt dot"
    "er time wastes life thou preventst"
    "l deeds better life provide public"
    "way threw savd life saying        "
    "hs vowd chaste life keep came trip"
```

Specify Source Text

Specify source text to display context.

Load the `sonnets.txt` data and split it into separate documents.

```
txt = extractFileText("sonnets.txt");
paragraphs = split(txt,[newline newline]);
```

Extract the sonnets from `paragraphs`. The first sonnet is the fifth element of paragraphs, and the remaining sonnets appear in every second element afterward.

```
sonnets = paragraphs(5:2:end);
documents = tokenizedDocument(sonnets);
```

Normalize the text, then search for the word "life".

```
documentsNormalized = normalizeWords(documents);
T = context(documentsNormalized,"life")
```

```
T=23×3 table
                            Context                            Document    Word
```

```
    ________________________________________________________    ________    ____

    "sum'st thy self in singl life ? ah ! if thou issueless"        9        18 
    " : so should the line of life that life repair , which"       16        73 
    "ld the line of life that life repair , which thi , tim"       16        75 
    "s a tomb which hide your life , and show not half your"       17        34 
    " live thi , and thi give life to thee .               "       18       128 
    "ssi of love to thee , my life , be made of four , with"       45        53 
    "eauti , though my lover' life : hi beauti shall in the"       63       100 
    " awai , to live a second life on second head ; er beau"       68        59 
    "t your love even with my life decai ; lest the wise wo"       71       118 
    "shall carri me awai , my life hath in thi line some in"       74        18 
    "ast but lost the dreg of life , the prei of worm , my "       74        83 
    "to my thought as food to life , or as sweet-season'd s"       75        10 
    "ur name from henc immort life shall have , though i , "       81        42 
    " , when other would give life , and bring a tomb . the"       83       108 
    "a tomb . there live more life in on of your fair ey th"       83       118 
    "yself awai , for term of life thou art assur mine ; an"       92        13 
      ⋮
```

Since the words are normalized, the contexts may not be easy to read. To view the contexts using the original text data, specify the source text using the `'Source'` option.

```
T = context(documentsNormalized,"life",'Source',sonnets)
```

```
T=23×3 table
                            Context                             Document    Word
    ________________________________________________________    ________    ____

    "um'st thy self in single life? Ah! if thou issueless s"        9        18 
    ": So should the lines of life that life repair, Which "       16        73 
    "d the lines of life that life repair, Which this, Time"       16        75 
    " a tomb Which hides your life, and shows not half your"       17        34 
    "ves this, and this gives life to thee.                "       18       128 
    "assy of love to thee, My life, being made of four, wit"       45        53 
    "eauty, though my lover's life: His beauty shall in the"       63       100 
    "n away, To live a second life on second head; Ere beau"       68        59 
    "t your love even with my life decay; Lest the wise wor"       71       118 
    " shall carry me away, My life hath in this line some i"       74        18 
    "st but lost the dregs of life, The prey of worms, my b"       74        83 
    "o my thoughts as food to life, Or as sweet-season'd sh"       75        10 
    "name from hence immortal life shall have, Though I, on"       81        42 
    ", When others would give life, and bring a tomb. There"       83       108 
    "a tomb. There lives more life in one of your fair eyes"       83       118 
    "hyself away, For term of life thou art assured mine; A"       92        13 
      ⋮
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`word` — Word to find
string scalar | character vector | scalar cell array

Word to find in context, specified as a string scalar, character vector, or scalar cell array containing a character vector.

Data Types: `char` | `string` | `cell`

`ngram` — N-gram to find
string array | cell array of character vectors

N-gram to find in context, specified as a string array or cell array of character vectors.

`ngram` has size `1`-by-`N` , where `N` is the number of words in the n-gram. The value of `ngram(j)` is the `j`th word of the n-gram.

The function ignores trailing empty strings in `ngram`.

Data Types: `string` | `cell`

`contextLength` — Context length
25 (default) | positive integer

Context length, specified as a positive integer.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Solver','avb'` specifies to use approximate variational Bayes as the solver.

`Source` — Source text
string array | cell array of character vectors

Source text, specified as the comma-separated pair consisting of `'Source'` and a string array or a cell array of character vectors. If the input documents are preprocessed, and you have the source text, then you can use this option to make the output more readable.

The source text must be the same size as `documents`.

`IgnoreCase` — Option to ignore case
`false` (default) | `true`

Option to ignore case, specified as the comma-separated pair consisting of `'IgnoreCase'` and one of the following:

- `false` – search for occurrences that match the word or n-gram exactly.
- `true` – search for occurrences that match the word or n-gram ignoring case.

Output Arguments

`T` — Table of contexts
table

Table of contexts with these columns:

`Context`	String containing the queried word or n-gram in context
`Document`	Numeric index of the document containing the word or n-gram
`Word`	Numeric indices of the word or n-gram in the document

See Also
`doc2cell` | `doclength` | `joinWords` | `string` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

correctSpelling

Correct spelling of words

Syntax

```
updatedDocuments = correctSpelling(documents)

updatedWords = correctSpelling(words)
updatedWords = correctSpelling(words,'Language',language)

[ ___ ,unknownWords] = correctSpelling( ___ )
 ___  = correctSpelling( ___ ,Name,Value)
```

Description

Use `correctSpelling` to correct spelling of words in string arrays or documents.

The function supports English, German, and Korean text.

`updatedDocuments = correctSpelling(documents)` corrects the spelling of the words in the `tokenizedDocument` array `documents`.

`updatedWords = correctSpelling(words)` corrects the spelling of the words in the `string` vector `words`.

`updatedWords = correctSpelling(words,'Language',language)` also specifies the language of the words in the `string` vector `words`.

`[ ___ ,unknownWords] = correctSpelling( ___ )` also returns a vector of words in the input that were not found in the dictionary and for which no suggestion was found.

`___ = correctSpelling( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Correct Spelling of Words in Documents

Create a tokenized document array.

```
str = [
    "A documnent containing some misspelled worrds."
    "Another documnent cntaining typos."];
documents = tokenizedDocument(str);
```

Correct the spelling of the words in the documents using the `correctSpelling` function.

```
updatedDocuments = correctSpelling(documents)
```

```
updatedDocuments = 
  2x1 tokenizedDocument:
    7 tokens: A document containing some misspelled words .
    5 tokens: Another document containing typos .
```

Correct Spelling of Words in String Array

Create a string array of words.

```
words = ["A" "strng" "array" "containing" "misspelled" "worrds" "."];
```

Correct the spelling of the words in the string array using the `correctSpelling` function.

```
updatedWords = correctSpelling(words)
```

```
updatedWords = 1x7 string
  Columns 1 through 6

    "A"    "string"    "array"    "containing"    "misspelled"    "words"

  Column 7

    "."
```

Specify Known Words

Create a tokenized document array.

```
str = [
    "Analyze text data using MATLAB."
    "Another documnent cntaining typos."];
documents = tokenizedDocument(str);
```

Correct the spelling of the words in the documents using the `correctSpelling` function.

```
updatedDocuments = correctSpelling(documents)
```

```
updatedDocuments = 
  2x1 tokenizedDocument:

    7 tokens: Analyze text data using MAT LAB .
    5 tokens: Another document containing typos .
```

Notice that the word "MATLAB" gets split into the two words "MAT" and "LAB".

Correct the spelling of the documents and specify "MATLAB" as a known word using the `'KnownWords'` option.

```
updatedDocuments = correctSpelling(documents,'KnownWords',"MATLAB")
```

```
updatedDocuments = 
  2x1 tokenizedDocument:
    6 tokens: Analyze text data using MATLAB .
    5 tokens: Another document containing typos .
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

`language` — Word language
`'en'` | `'de'` | `'ko'`

Word language, specified as one of the following:

- `'en'` – English language
- `'de'` – German language
- `'ko'` – Korean language

If you do not specify language, then the software detects the language automatically.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `correctSpelling(documents,'KnownWords',["MathWorks" "MATLAB"])` corrects the spelling of the words in `documents` and treats the words "MathWorks" and "MATLAB" as correctly spelled words.

`KnownWords` — Words to be treated as correct
`[]` (default) | string array | cell array of character vectors

Words to be treated as correct, specified as the comma-separated pair consisting of `'KnownWords'` and a string array or a cell array of character vectors.

If you specify a list of known words, then these words remain unchanged when the function corrects spelling. The software may also substitute misspelled words with words from the list of known words.

Example: `["MathWorks" "MATLAB"]`

Data Types: `char` | `string` | `cell`

ExtensionDictionary — Hunspell extension dictionary file
'' (default) | file path

Hunspell extension dictionary file (also known as personal dictionary file), specified as the comma-separated pair consisting of 'ExtensionDictionary' and a file path of a Hunspell extension dictionary file.

A Hunspell extension dictionary file is a .dic file containing the number of words in the dictionary followed by a list of the words in the following format:

```
word1/affixWord1
word2/affixWord2
...
wordN/affixWordN
*forbiddenWord1
*forbiddenWord2
...
*forbiddenWordM
```

where:

- word1, word2, ..., wordN is a list words to extend the Hunspell dictionary with.
- affixWord1, affixWord2, ..., affixWordN (optional) indicate words in the Hunspell dictionary that share affixes. Indicate affixes by concatenating them to the corresponding word with a forward slash (/). For example, the entry exxxtreme/extreme indicates that affixes that apply to the word "extreme" also apply to the custom word "exxxtreme".
- forbiddenWord1, forbiddenWord2, ..., forbiddenWordN is a list of forbidden words to use for spelling correction. Indicate forbidden words using an asterisk (*).

The entries in the Hunspell extension dictionary file can appear in any order.

For example, to create a Hunspell extension dictionary file specifying:

- The words "MathWorks", "MATLAB", and "exxxtreme".
- The affixes that apply to the word "extreme" also apply to the word "exxxtreme".
- The word "MATLOB" is a forbidden word.

use:

```
MathWorks
MATLAB
exxxtreme/extreme
*MATLOB
```

For an example showing how to create Hunspell extension dictionary files, see “Create Extension Dictionary for Spelling Correction”. For more information about the options of Hunspell dictionary files, see https://manpages.ubuntu.com/manpages/trusty/en/man4/hunspell.4.html.

Data Types: char | string

Dictionary — Hunspell dictionary file
'' (default) | file path

Hunspell dictionary file, specified as the comma-separated pair consisting of 'Dictionary' and a file path of a Hunspell dictionary file.

A Hunspell dictionary file is a `.dic` file containing the number of words in the dictionary followed by a list of the words in the following format:

```
N
word1/flags1
word2/flags2
...
wordN/flagsN
```

where `N` is the number of words in the dictionary file, `word1`, `word2`, ..., `wordN` are the `N` words in the dictionary, and `flags1`, ..., `flagsN` specify optional flags corresponding to the words `word1`, `word2`, ..., `wordN`, respectively. Use flags to specify word attributes, for example affixes. To specify a Hunspell affix file, use the `'Affixes'` option.

For example, a to create a Hunspell dictionary file containing the 4 words `"MathWorks"`, `"MATLAB"`, `"correctSpelling"`, and `"tokenizedDocument"`, use:

```
4
MathWorks
MATLAB
correctSpelling
tokenizedDocument
```

For more information about the options of Hunspell dictionary files, see https://manpages.ubuntu.com/manpages/trusty/en/man4/hunspell.4.html.

Data Types: `char` | `string`

`Affixes` — Hunspell affix file
`''` (default) | file path

Hunspell affix file, specified as the comma-separated pair consisting of `'Affixes'` and a file path of a Hunspell affix file.

A Hunspell affix file is a `.aff` file containing the number of words in the dictionary followed by a list of the words in the following format:

```
option1 values1
option2 values2
...
optionM valuesM
```

where `M` is the number of options in the affix file, `option1`, `option2`, ..., `optionM` are the `M` options, and `values1`, ..., `valuesN` specify the values corresponding to the options `option1`, `option2`, ..., `optionM`, respectively. Use these options to specify affixes.

Prefixes

To define a prefix rule, use the `PFX` option with the format:

```
PFX flag crossProduct K
PFX flag stripping1 prefix1 condition1
...
PFX flag strippingK prefixK conditionK
```

where the values:

- `flag` corresponds to the flags used in the Hunspell dictionary file.
- `crossProduct` indicates whether prefixes and suffixes can be mixed, specified as `Y` or `N`.
- `K` is the number of prefixes defined for the specified flag.
- `stripping1`, `stripping2`, ..., `strippingK` indicate characters to be stripped from the word when applying prefix. If the stripping value is `0`, then no stripping takes place.
- `prefix1`, `prefix2`, ..., `prefixK` specify the prefixes to use.
- `condition1`, `condition2`, ..., `conditionK` specify the optional conditions for which to apply the prefixes `prefix1`, `prefix2`, ..., `prefixK`, respectively. For the trivial condition, specify `"."`.

Suffixes

To define a suffix rule, use the `SFX` option with the format:

```
SFX flag crossProduct K
SFX flag stripping1 suffix1 condition1
...
SFX flag strippingK suffixK conditionK
```

where `suffix1`, `suffix2`, ..., `suffixK` specify the prefixes to use, and the flag, cross product, `K`, stripping, and condition values are the same as the prefix format.

Example

Create a Hunspell affix file defining the following affix rules:

- Flag A:
  - prefix words with `"re"`
- Flag B:
  - suffix words not ending with `"y"` with `"ed"`.
  - suffix words ending with `"y"` with `"ied"`, removing `"y"`.

use the Hunspell affix file:

```
PFX A Y 1
PFX A 0 re .

SFX B Y 1
SFX B 0 ed [^y]
SFX B y ied y
```

To use these flags in a Hunspell dictionary file, append the appropriate flags to the words using the `"/"`. For each word, you can specify multiple flags. For example, to specify a dictionary file containing:

- The words `"ptest"` and `"ptry"`.
- For the word `"ptest"` only, also include the prefix `"re"` using flag `A`.
- For both words, also include the suffixes `"ed"` or `"ied"` where appropriate using flag `B`

For more information about the options of Hunspell affix files, see https://manpages.ubuntu.com/manpages/trusty/en/man4/hunspell.4.html.

Data Types: `char` | `string`

`RetokenizeMethod` — Method to retokenize documents
`'split'` (default) | `'none'`

Method to retokenize documents, specified as the comma-separated pair consisting of `'RetokenizeMethod'` and one of the following:

- `'split'` – Correct spelling by splitting tokens. For example, split the incorrectly spelled token `"twowords"` into the correctly spelled tokens `"two"` and `"words"`.
- `'none'` – Do not split tokens for spelling correction.

Output Arguments

`updatedDocuments` — Corrected documents
`tokenizedDocument` array

Corrected documents, returned as a `tokenizedDocument` array. If the `'RetokenizeMethod'` option is `'split'`, then the number of words in each updated document may be different to the corresponding input document.

If there are multiple candidates for corrected words, then the function automatically selects a single word for correction.

`updatedWords` — Corrected words
string vector

Corrected words, returned as a string vector. If the `'RetokenizeMethod'` option is `'split'`, then the number of updated words may be different the number of input words.

If there are multiple candidates for corrected words, then the function automatically selects a single word for correction.

`unknownWords` — Unknown words
string vector

Unknown words, returned as a string vector. The string vector `unknownWords` contains the input words that are not in the spelling correction dictionary and for which no suggestions are found.

See Also

`editDistance` | `editDistanceSearcher` | `tokenizedDocument`

Topics
"Correct Spelling in Documents"
"Create Extension Dictionary for Spelling Correction"
"Create Custom Spelling Correction Function Using Edit Distance Searchers"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"

Introduced in R2020a

corpusLanguage

Detect language of text

Syntax

```
language = corpusLanguage(str)
```

Description

Use `corpusLanguage` to detect language of text.

The function supports English, Japanese, German, and Korean text.

`language = corpusLanguage(str)` detects the language of the text in `str`.

Examples

Detect Language of Text

Detect the language of a string array of text.

```
str = [
    "恋の悩みで　苦しむ。"
    "空の星が輝きを増している。"];
language = corpusLanguage(str)
```

```
language = 
'ja'
```

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

Example: `["An example of a short sentence."; "A second short sentence."]`

Data Types: `string` | `char` | `cell`

Output Arguments

`language` — Detected language
`'en'` | `'ja'` | `'de'` | `'ko'`

Detected language, returned as one of the following:

- `'en'` – Detected English text

- `'ja'` – Detected Japanese text
- `'de'` – Detected German text
- `'ko'` – Detected Korean text

See Also

`abbreviations` | `addLanguageDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `addSentenceDetails` | `splitSentences` | `tokenDetails` | `tokenizedDocument` | `topLevelDomains`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Visualize Text Data Using Word Clouds"
"Language Considerations"
"Japanese Language Support"
"German Language Support"

Introduced in R2018b

cosineSimilarity

Document similarities with cosine similarity

Syntax

```
similarities = cosineSimilarity(documents)
similarities = cosineSimilarity(documents,queries)

similarities = cosineSimilarity(bag)
similarities = cosineSimilarity(bag,queries)

similarities = cosineSimilarity(M)
similarities = cosineSimilarity(M1,M2)
```

Description

`similarities = cosineSimilarity(documents)` returns the pairwise cosine similarities for the specified documents using the tf-idf matrix derived from their word counts. The score in `similarities(i,j)` represents the similarity between `documents(i)` and `documents(j)`.

`similarities = cosineSimilarity(documents,queries)` returns similarities between `documents` and `queries` using tf-idf matrices derived from the word counts in `documents`. The score in `similarities(i,j)` represents the similarity between `documents(i)` and `queries(j)`.

`similarities = cosineSimilarity(bag)` returns pairwise similarities for the documents encoded by the specified bag-of-words or bag-of-n-grams model using the tf-idf matrix derived from the word counts in `bag`. The score in `similarities(i,j)` represents the similarity between the `i`th and `j`th documents encoded by `bag`.

`similarities = cosineSimilarity(bag,queries)` returns similarities between the documents encoded by the bag-of-words or bag-of-n-grams model `bag` and `queries` using tf-idf matrices derived from the word counts in `bag`. The score in `similarities(i,j)` represents the similarity between the `i`th document encoded by `bag` and `queries(j)`.

`similarities = cosineSimilarity(M)` returns similarities for the data encoded in the row vectors of the matrix `M`. The score in `similarities(i,j)` represents the similarity between `M(i,:)` and `M(j,:)`.

`similarities = cosineSimilarity(M1,M2)` returns similarities between the documents encoded in the matrices `M1` and `M2`. The score in `similarities(i,j)` corresponds to the similarity between `M1(i,:)` and `M2(j,:)`.

Examples

Similarity Between Documents

Create an array of tokenized documents.

```
textData = [
    "the quick brown fox jumped over the lazy dog"
    "the fast brown fox jumped over the lazy dog"
    "the lazy dog sat there and did nothing"
    "the other animals sat there watching"];
documents = tokenizedDocument(textData)
```

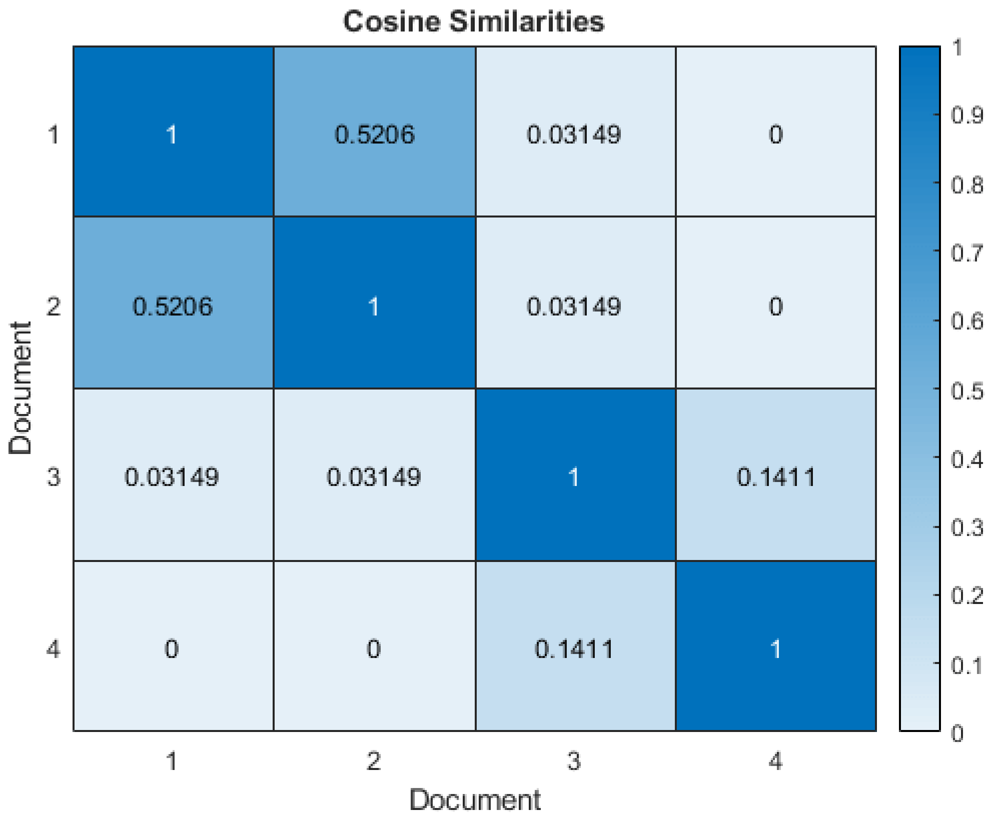
```
documents = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    9 tokens: the fast brown fox jumped over the lazy dog
    8 tokens: the lazy dog sat there and did nothing
    6 tokens: the other animals sat there watching
```

Calculate the similarities between them using the `cosineSimilarity` function. The output is a sparse matrix.

```
similarities = cosineSimilarity(documents);
```

Visualize the similarities between the documents in a heat map.

```
figure
heatmap(similarities);
xlabel("Document")
ylabel("Document")
title("Cosine Similarities")
```



Scores close to one indicate strong similarity. Scores close to zero indicate weak similarity.

Similarity to Query

Create an array of input documents.

```
str = [
    "the quick brown fox jumped over the lazy dog"
    "the fast fox jumped over the lazy dog"
    "the dog sat there and did nothing"
    "the other animals sat there watching"];
documents = tokenizedDocument(str)
```

```
documents = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    8 tokens: the fast fox jumped over the lazy dog
    7 tokens: the dog sat there and did nothing
    6 tokens: the other animals sat there watching
```

Create an array of query documents.

```
str = [
    "a brown fox leaped over the lazy dog"
    "another fox leaped over the dog"];
queries = tokenizedDocument(str)
```

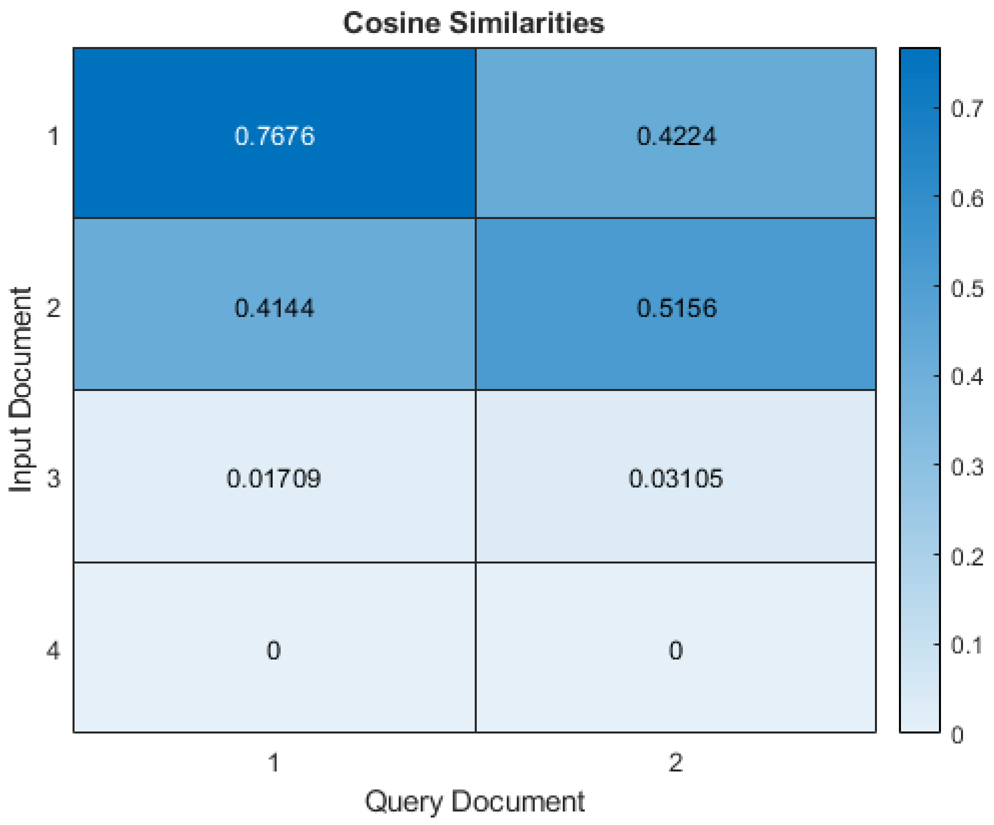
```
queries = 
  2x1 tokenizedDocument:

    8 tokens: a brown fox leaped over the lazy dog
    6 tokens: another fox leaped over the dog
```

Calculate the similarities between input and query documents using the `cosineSimilarity` function. The output is a sparse matrix.

```
similarities = cosineSimilarity(documents,queries);
```

Visualize the similarities of the documents in a heat map.

```
figure
heatmap(similarities);
xlabel("Query Document")
ylabel("Input Document")
title("Cosine Similarities")
```



Scores close to one indicate strong similarity. Scores close to zero indicate weak similarity.

Document Similarities Using Bag-of-Words Model

Create a bag-of-words model from the text data in `sonnets.csv`.

```
filename = "sonnets.csv";
tbl = readtable(filename,'TextType','string');
textData = tbl.Sonnet;
documents = tokenizedDocument(textData);
bag = bagOfWords(documents)
```

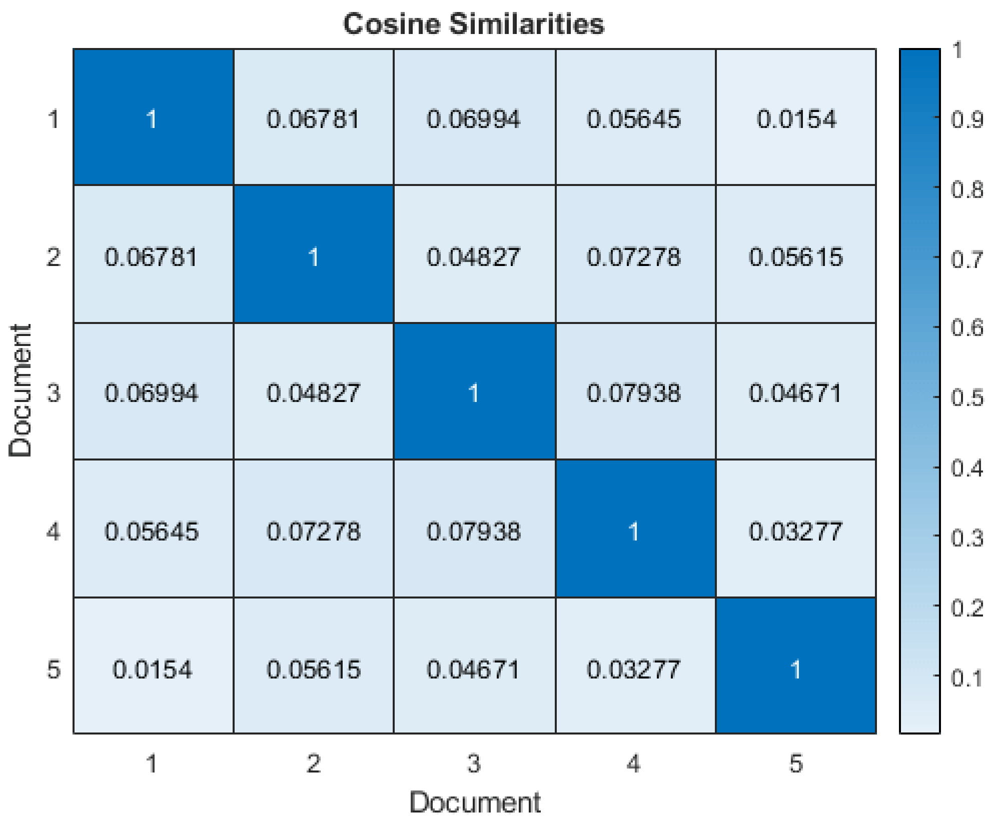
```
bag = 
  bagOfWords with properties:

          Counts: [154x3527 double]
      Vocabulary: [1x3527 string]
        NumWords: 3527
    NumDocuments: 154
```

Calculate similarities between the sonnets using the `cosineSimilarity` function. The output is a sparse matrix.

```
similarities = cosineSimilarity(bag);
```

Visualize the similarities of the first five documents in a heat map.

```
figure
heatmap(similarities(1:5,1:5));
xlabel("Document")
ylabel("Document")
title("Cosine Similarities")
```



Scores close to one indicate strong similarity. Scores close to zero indicate weak similarity.

Similarities Within Word Count Matrix

For bag-of-words input, the `cosineSimilarity` function calculates the cosine similarity using the tf-idf matrix derived from the model. To compute the cosine similarities on the word count vectors directly, input the word counts to the `cosineSimilarity` function as a matrix.

Create a bag-of-words model from the text data in `sonnets.csv`.

```
filename = "sonnets.csv";
tbl = readtable(filename,'TextType','string');
textData = tbl.Sonnet;
documents = tokenizedDocument(textData);
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:
        Counts: [154x3527 double]
    Vocabulary: [1x3527 string]
      NumWords: 3527
  NumDocuments: 154
```

Get the matrix of word counts from the model.

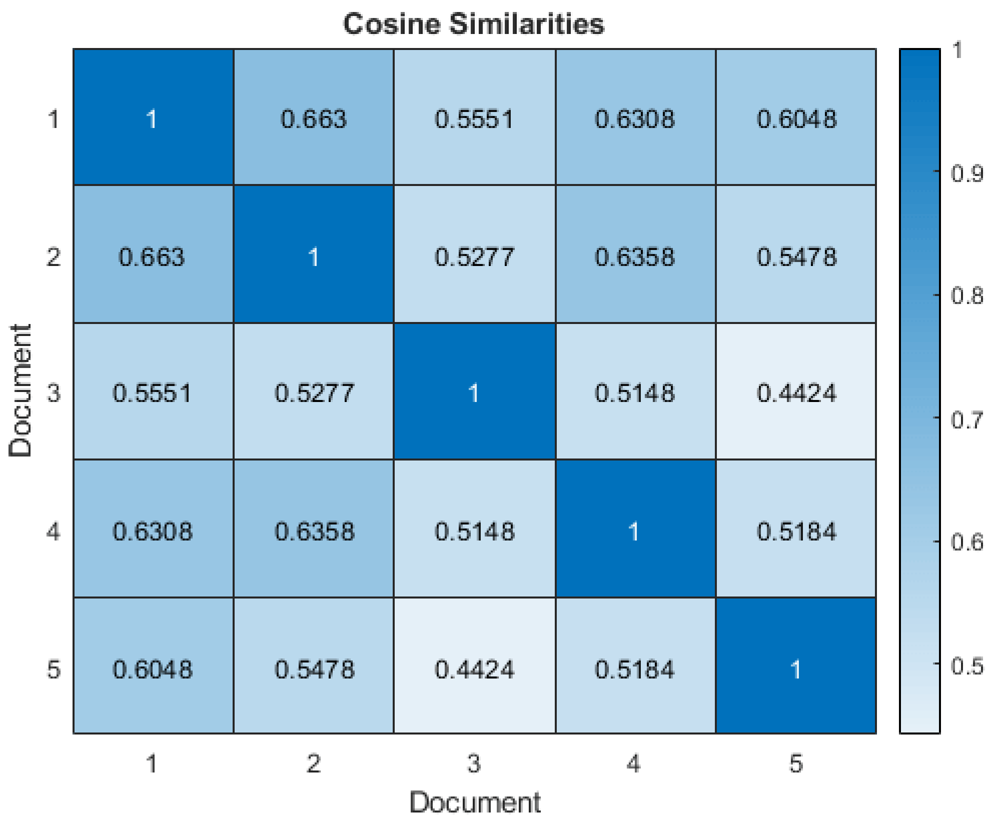
```
M = bag.Counts;
```

Calculate the cosine document similarities of the word count matrix using the `cosineSimilarity` function. The output is a sparse matrix.

```
similarities = cosineSimilarity(M);
```

Visualize the similarities of the first five documents in a heat map.

```
figure
heatmap(similarities(1:5,1:5));
xlabel("Document")
ylabel("Document")
title("Cosine Similarities")
```



	1	2	3	4	5
1	1	0.663	0.5551	0.6308	0.6048
2	0.663	1	0.5277	0.6358	0.5478
3	0.5551	0.5277	1	0.5148	0.4424
4	0.6308	0.6358	0.5148	1	0.5184
5	0.6048	0.5478	0.4424	0.5184	1

Scores close to one indicate strong similarity. Scores close to zero indicate weak similarity.

Input Arguments

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

`bag` — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`queries` — Set of query documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Set of query documents, specified as one of the following:

- A `tokenizedDocument` array
- A 1-by-*N* string array representing a single document, where each element is a word
- A 1-by-*N* cell array of character vectors representing a single document, where each element is a word

To compute term frequency and inverse document frequency statistics, the function encodes `queries` using a bag-of-words model. The model it uses depends on the syntax you call it with. If your syntax specifies the input argument `documents`, then it uses `bagOfWords(documents)`. If your syntax specifies `bag`, then the function encodes `queries` using `bag` then uses the resulting tf-idf matrix.

`M` — Input data
matrix

Input data, specified as a matrix. For example, `M` can be a matrix of word or n-gram counts or a tf-idf matrix.

Data Types: `double`

Output Arguments

`similarities` — Cosine similarity scores
sparse matrix

Cosine similarity scores, returned as a sparse matrix:

- Given a single array of tokenized documents, `similarities` is a *N*-by-*N* symmetric matrix, where `similarities(i,j)` represents the similarity between `documents(i)` and `documents(j)`, and N is the number of input documents.
- Given an array of tokenized documents and a set of query documents, `similarities` is an *N1*-by-*N2* matrix, where `similarities(i,j)` represents the similarity between `documents(i)` and the `j`th query document, and *N1* and *N2* represents the number of documents in `documents` and `queries`, respectively.

- Given a single bag-of-words or bag-of-n-grams model, `similarities` is a `bag.NumDocuments`-by-`bag.NumDocuments` symmetric matrix, where `similarities(i,j)` represents the similarity between the `i`th and `j`th documents encoded by `bag`.
- Given a bag-of-words or bag-of-n-grams models and a set of query documents, `similarities` is a `bag.NumDocuments`-by-*N2* matrix, where `similarities(i,j)` represents the similarity between the `i`th document encoded by `bag` and the `j`th document in `queries`, and *N2* corresponds to the number of documents in `queries`.
- Given a single matrix, `similarities` is a `size(M,1)`-by-`size(M,1)` symmetric matrix, where `similarities(i,j)` represents the similarity between `M(i,:)` and `M(j,:)`.
- Given two matrices, `similarities` is an `size(M1,1)`-by-`size(M2,1)` matrix, where `similarities(i,j)` represents the similarity between `M1(i,:)` and `M2(j,:)`.

See Also

`bleuEvaluationScore` | `bm25Similarity` | `extractSummary` | `lexrankScores` | `mmrScores` | `rougeEvaluationScore` | `textrankScores` | `tokenizedDocument`

Topics

"Sequence-to-Sequence Translation Using Attention"

Introduced in R2020a

decodeHTMLEntities

Convert HTML and XML entities into characters

Syntax

```
newStr = decodeHTMLEntities(str)
```

Description

`newStr = decodeHTMLEntities(str)` replaces HTML and XML character entities and numeric character references in the elements of `str` with their Unicode equivalent.

Examples

Replace HTML Entities with Unicode

Replace HTML character entities with their Unicode equivalent.

```
str = ["&lt;&gt;" "R&amp;D"];
newStr = decodeHTMLEntities(str)
```

```
newStr = 1x2 string
    "<>"    "R&D"
```

Replace HTML numeric character references with their Unicode equivalent. Unicode character with hex code `&#x20` is a space.

```
str = "R&#x20;D";
newStr = decodeHTMLEntities(str)
```

```
newStr = 
"R D"
```

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

Example: `["An example of a short sentence."; "A second short sentence."]`

Data Types: `string` | `char` | `cell`

Output Arguments

`newStr` — Output text
string array | character vector | cell array of character vectors

Output text, returned as a string array, a character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

See Also

`erasePunctuation` | `eraseTags` | `eraseURLs` | `lower` | `tokenizedDocument` | `upper`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

doclength

Length of documents in document array

Syntax

```
N = doclength(documents)
```

Description

`N = doclength(documents)` returns the number of tokens in each document in `documents`.

Examples

Find Number of Words in Documents

Find the number of words in an array of tokenized documents. Erase the punctuation characters so they do not get counted as words.

```
str = [ ...
    "An example of a short sentence."
    "A second short sentence."];
documents = tokenizedDocument(str)
```

```
documents = 
  2x1 tokenizedDocument:

    7 tokens: An example of a short sentence .
    5 tokens: A second short sentence .
```

```
documents = erasePunctuation(documents)
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: An example of a short sentence
    4 tokens: A second short sentence
```

```
N = doclength(documents)
```

```
N = 2×1

     6
     4
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`N` — Document lengths
vector of nonnegative integers

Document lengths, returned as a vector of nonnegative integers. The size of `N` is the same as the size of `documents`.

See Also

`context` | `doc2cell` | `joinWords` | `string` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

doc2cell

Convert documents to cell array of string vectors

Syntax

```
C = doc2cell(documents)
```

Description

`C = doc2cell(documents)` converts a `tokenizedDocument` array to a cell array. The entries of `C` are string arrays containing the corresponding words in each document.

Examples

Convert Document Array to Cell Array

Convert a `tokenizedDocument` array to a cell array of string vectors.

```
documents = tokenizedDocument([ ...
    "an example of a short sentence" ...
    "a second short sentence"])
```

```
documents = 
  1x2 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

```
C = doc2cell(documents)
```

```
C=1×2 cell array
    {1x6 string}    {1x4 string}
```

View the first element of the cell array.

```
C{1}
```

```
ans = 1x6 string
    "an"    "example"    "of"    "a"    "short"    "sentence"
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`C` — Output cell array
cell array of string vectors

Output cell array of string vectors. Each element of `C` is a string vector containing the words of the corresponding document.

See Also

`context` | `doclength` | `joinWords` | `string` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

doc2sequence

Convert documents to sequences for deep learning

Syntax

```
sequences = doc2sequence(enc,documents)
sequences = doc2sequence(emb,documents)
sequences = doc2sequence( ___ ,Name,Value)
```

Description

`sequences = doc2sequence(enc,documents)` returns a cell array of the numeric indices of the words in `documents` given by the word encoding `enc`. Each element of `sequences` is a vector of the indices of the words in the corresponding document.

`sequences = doc2sequence(emb,documents)` returns a cell array of the embedding vectors of the words in `documents` given by the word embedding `emb`. Each element of `sequences` is a matrix of the embedding vectors of the words in the corresponding document.

`sequences = doc2sequence( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Convert Documents to Sequences of Word Indices

Load the factory reports data and create a `tokenizedDocument` array.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
textData = data.Description;
documents = tokenizedDocument(textData);
```

Create a word encoding.

```
enc = wordEncoding(documents);
```

Convert the documents to sequences of word indices.

```
sequences = doc2sequence(enc,documents);
```

View the sizes of the first 10 sequences. Each sequence is a 1-by-*S* vector, where *S* is the number of word indices in the sequence. Because the sequences are padded, *S* is constant.

```
sequences(1:10)
```

```
ans=10×1 cell array
    {[         0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10]}
    {[  0 0 0 0 0 0 11 12 13 14 15 2 16 17 18 19 10]}
    {[    0 0 0 0 0 0 20 2 21 22 7 23 24 25 7 26 10]}
    {[        0 0 0 0 0 0 0 0 0 0 0 27 28 6 7 18 10]}
    {[        0 0 0 0 0 0 0 0 0 0 0 0 29 30 7 31 10]}
    {[    0 0 0 0 0 0 0 32 33 6 7 34 35 36 37 38 10]}
    {[      0 0 0 0 0 0 0 0 0 39 40 36 41 6 7 42 10]}
    {[    0 0 0 0 0 0 0 0 43 44 22 45 46 47 7 48 10]}
    {[      0 0 0 0 0 0 0 0 0 0 49 50 17 7 51 48 10]}
    {[0 0 0 0 52 8 53 36 54 55 56 57 58 59 22 60 10]}
```

Convert Documents to Sequences of Word Vectors

Convert an array of tokenized documents to sequences of word vectors using a pretrained word embedding.

Load a pretrained word embedding using the `fastTextWordEmbedding` function. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding;
```

Load the factory reports data and create a `tokenizedDocument` array.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
textData = data.Description;
documents = tokenizedDocument(textData);
```

Convert the documents to sequences of word vectors using `doc2sequence`. The `doc2sequence` function, by default, left-pads the sequences to have the same length. When converting large collections of documents using a high-dimensional word embedding, padding can require large amounts of memory. To prevent the function from padding the data, set the `'PaddingDirection'` option to `'none'`. Alternatively, you can control the amount of padding using the `'Length'` option.

```
sequences = doc2sequence(emb,documents,'PaddingDirection','none');
```

View the sizes of the first 10 sequences. Each sequence is *D*-by-*S* matrix, where *D* is the embedding dimension, and *S* is the number of word vectors in the sequence.

```
sequences(1:10)
```

```
ans=10×1 cell array
    {300×10 single}
    {300×11 single}
    {300×11 single}
    {300×6  single}
    {300×5  single}
    {300×10 single}
    {300×8  single}
    {300×9  single}
    {300×7  single}
    {300×13 single}
```

Pad or Truncate Sequences to Specified Length

Convert a collection of documents to sequences of word vectors using a pretrained word embedding, and pad or truncate the sequences to a specified length.

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding;
```

Load the factory reports data and create a `tokenizedDocument` array.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
textData = data.Description;
documents = tokenizedDocument(textData);
```

Convert the documents to sequences of word vectors. Specify to left-pad or truncate the sequences to have length 100.

```
sequences = doc2sequence(emb,documents,'Length',100);
```

View the sizes of the first 10 sequences. Each sequence is *D*-by-*S* matrix, where *D* is the embedding dimension, and *S* is the number of word vectors in the sequence (the sequence length). Because the sequence length is specified, *S* is constant.

```
sequences(1:10)
```

```
ans=10×1 cell array
    {300×100 single}
    {300×100 single}
    {300×100 single}
    {300×100 single}
    {300×100 single}
    {300×100 single}
    {300×100 single}
    {300×100 single}
    {300×100 single}
    {300×100 single}
```

Input Arguments

emb — Input word embedding
`wordEmbedding` object

Input word embedding, specified as a `wordEmbedding` object.

enc — Input word encoding
`wordEncoding` object

Input word encoding, specified as a `wordEncoding` object.

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Length','shortest'` truncates the sequences to have the same length as the shortest sequence.

`UnknownWord` — Unknown word behavior

`'discard'` (default) | `'nan'`

Unknown word behavior, specified as the comma-separated pair consisting of `'UnknownWord'` and one of the following:

- `'discard'` – If a word is not in the input map, then discard it.
- `'nan'` – If a word is not in the input map, then return a `NaN` value.

Tip If you are creating sequences for training a deep learning network with a word embedding, use `'discard'`. Do not use sequences with `NaN` values, because doing so can propagate errors through the network.

`PaddingDirection` — Padding direction

`'left'` (default) | `'right'` | `'none'`

Padding direction, specified as the comma-separated pair consisting of `'PaddingDirection'` and one of the following:

- `'left'` – Pad sequences on the left.
- `'right'` – Pad sequences on the right.
- `'none'` – Do not pad sequences.

Tip When converting large collections of data using a high-dimensional word embedding, padding can require large amounts of memory. To prevent the function from adding too much padding, set the `'PaddingDirection'` option to `'none'` or set `'Length'` to a smaller value.

`PaddingValue` — Padding value

0 (default) | numeric scalar

Padding value, specified as the comma-separated pair consisting of `'PaddingValue'` and a numeric scalar. Do not pad sequences with `NaN`, because doing so can propagate errors through the network.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Length` — Sequence length

`'longest'` (default) | `'shortest'` | positive integer

Sequence length, specified as the comma-separated pair consisting of `'Length'` and one of the following:

- `'longest'` – Pad sequences to have the same length as the longest sequence.
- `'shortest'` – Truncate sequences to have the same length as the shortest sequence.
- Positive integer – Pad or truncate sequences to have the specified length. The function truncates the sequences on the right.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

Output Arguments

`sequences` — Output sequences
cell array

Output sequences, returned as a cell array.

For word embedding input, the *i*th element of `sequences` is a matrix of the word vectors corresponding to the *i*th input document.

For word encoding input, the *i*th element of `sequences` is a vector of the word encoding indices corresponding to the *i*th input document.

Tips

- When converting large collections of data using a high-dimensional word embedding, padding can require large amounts of memory. To prevent the function from adding too much padding, set the `'PaddingDirection'` option to `'none'` or set `'Length'` to a smaller value.

See Also

`fastTextWordEmbedding` | `ind2word` | `isVocabularyWord` | `tokenizedDocument` | `trainWordEmbedding` | `vec2word` | `word2ind` | `word2vec` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding`

Topics

“Train a Sentiment Classifier”
“Classify Text Data Using Deep Learning”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2018b

docfun

Apply function to words in documents

Syntax

```
newDocuments = docfun(func,documents)
newDocuments = docfun(func,documents1,...,documentsN)
```

Description

`newDocuments = docfun(func,documents)` calls the function specified by the function handle `func` and passes elements of `documents` as a string vector of words.

- If `func` accepts exactly one input argument, then the words of `newDocuments(i)` are the output of `func(string(documents(i)))`.
- If `func` accepts two input arguments, then the words of `newDocuments(i)` are the output of `func(string(documents(i)),details)`, where `details` contains the corresponding token details output by `tokenDetails`.
- If `func` changes the number of words in the document, then `docfun` removes the token details from that document.

`docfun` does not perform the calls to function `func` in a specific order.

`newDocuments = docfun(func,documents1,...,documentsN)` calls the function specified by the function handle `func` and passes elements of `documents1,…,documentsN` as string vectors of words, where *N* is the number of inputs to the function `func`. The words of `newDocuments(i)` are the output of `func(string(documents1(i)),...,string(documentsN(i)))`.

Each of `documents1,…,documentsN` must be the same size.

Examples

Reverse Words in Documents

Apply `reverse` to each word in a document array.

```
documents = tokenizedDocument([ ...
    "an example of a short sentence"
    "a second short sentence"])

documents = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence


func = @reverse;
newDocuments = docfun(func,documents)
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: na elpmaxe fo a trohs ecnetnes
    4 tokens: a dnoces trohs ecnetnes
```

Specify Document Function with Multiple Inputs

Tag words by combining the words from one document array with another, using the string function `plus`.

Create the first `tokenizedDocument` array. Erase the punctuation and convert the text to lowercase.

```
str = [ ...
    "An example of a short sentence."
    "A second short sentence."];
str = erasePunctuation(str);
str = lower(str);
documents1 = tokenizedDocument(str)
```

```
documents1 = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

Create the second `tokenizedDocument` array. The documents have the same number of words as the corresponding documents in `documents1`. The words of `documents2` are POS tags for the corresponding words.

```
documents2 = tokenizedDocument([ ...
    "_det _noun _prep _det _adj _noun"
    "_det _adj _adj _noun"])
```

```
documents2 = 
  2x1 tokenizedDocument:

    6 tokens: _det _noun _prep _det _adj _noun
    4 tokens: _det _adj _adj _noun
```

```
func = @plus;
newDocuments = docfun(func,documents1,documents2)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: an_det example_noun of_prep a_det short_adj sentence_noun
    4 tokens: a_det second_adj short_adj sentence_noun
```

The output is not the same as calling `plus` on the documents directly.

```
plus(documents1,documents2)
```

```
ans = 
  2x1 tokenizedDocument:

    12 tokens: an example of a short sentence _det _noun _prep _det _adj _noun
     8 tokens: a second short sentence _det _adj _adj _noun
```

Input Arguments

`func` — Function handle
function handle

Function handle that accepts *N* string arrays as inputs and outputs a string array. `func` must accept `string(documents1(i)),...,string(documentsN(i))` as input.

Function handle to apply to words in documents. The function must have one of the following syntaxes:

- `newWords = func(words)`, where `words` is a string array of the words of a single document.
- `newWords = func(words,details)`, where `words` is a string array of the words of a single document, and `details` is the corresponding table of token details given by `tokenDetails`.
- `newWords = func(words1,...,wordsN)`, where `words1,...,wordsN` are string arrays of words.

Example: `@reverse`

Data Types: `function_handle`

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

See Also

`addPartOfSpeechDetails` | `addSentenceDetails` | `bagOfNgrams` | `bagOfWords` | `decodeHTMLEntities` | `lower` | `plus` | `regexprep` | `replace` | `tokenDetails` | `tokenizedDocument` | `upper`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Create Custom Spelling Correction Function Using Edit Distance Searchers”

Introduced in R2017b

editDistance

Find edit distance between two strings or documents

Syntax

```
d = editDistance(str1,str2)
d = editDistance(document1,document2)
d = editDistance( ___ ,Name,Value)
```

Description

`d = editDistance(str1,str2)` returns the lowest number of grapheme (Unicode term for human-perceived characters) insertions, deletions, and substitutions required to convert `str1` to `str2`.

`d = editDistance(document1,document2)` returns the lowest number of token insertions, deletions, and substitutions required to convert `document1` to `document2`.

`d = editDistance( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Edit Distance Between Two Strings

Find the edit distance between the strings `"Text analytics"` and `"Text analysis"`. The edit distance, by default, is the total number of grapheme insertions, deletions, and substitutions required to change one string to another.

```
str1 = "Text analytics";
str2 = "Text analysis";
```

Find the edit distance.

```
d = editDistance(str1,str2)
```

```
d = 2
```

This means changing the first string to the second requires two edits. For example:

1. Substitution – Substitute the character `"t"` to an `"s"`: `"Text analytics"` to `"Text analysics"`.
2. Deletion – Delete the character `"c"`: `"Text analysics"` to `"Text analysis"`.

Edit Distance Between Two Documents

Find the edit distance between two tokenized documents. For tokenized document input, the edit distance, by default, is the total number of token insertions, deletions, and substitutions required to change one document to another.

```
str1 = "It's time for breakfast.";
document1 = tokenizedDocument(str1);

str2 = "It's now time to sleep.";
document2 = tokenizedDocument(str2);
```

Find the edit distance.

```
d = editDistance(document1,document2)
```

```
d = 3
```

This means changing the first document to the second requires three edits. For example:

1. Insertion – Insert the word "`now`".
2. Substitution – Substitute the word "`for`" with "`to`".
3. Substitution – Substitute the word "`breakfast`" with "`sleep`".

Specify Cost Values

The `editDistance` function, by default, returns the lowest number of grapheme insertions, deletions, and substitutions required to change one string to another. To also include the swap action in the calculation, use the `'SwapCost'` option.

First, find the edit distance between the strings "`MATALB`" and "`MATLAB`".

```
str1 = "MATALB";
str2 = "MATLAB";
d = editDistance(str1,str2)
```

```
d = 2
```

One possible edit is:

1. Substitute the second "`A`" with "`L`": ("`MATALB`" to "`MATLLB`").
2. Substitute the second "`L`" with "`A`": ("`MATLLB`" to "`MATLAB`").

The default value for the swap cost (the cost of swapping two adjacent graphemes) is `Inf`. This means that swaps do not count towards the edit distance. To include swaps, set the `'SwapCost'` option to 1.

```
d = editDistance(str1,str2,'SwapCost',1)
```

```
d = 1
```

This means there is one action. For example, swap the adjacent characters "`A`" and "`L`".

Specify Custom Cost Function

To compute the edit distance between two words and specify that the edits are case-insensitive, specify a custom substitute cost function.

First, compute the edit distance between the strings "MATLAB" and "MathWorks".

```
d = editDistance("MATLAB","MathWorks")
```

```
d = 8
```

This means changing the first string to the second requires eight edits. For example:

1. Substitution – Substitute the character "A" with "a". ("MATLAB" to "MaTLAB")
2. Substitution – Substitute the character "T" with "t". ("MaTLAB" to "MatLAB")
3. Substitution – Substitute the character "L" with "h". ("MatLAB" to "MathAB")
4. Substitution – Substitute the character "A" with "W". ("MathAB" to "MathWB")
5. Substitution – Substitute the character "B" with "o". ("MathWB" to "MathWo")
6. Insertion – Insert the character "r". ("MathWo" to "MathWor")
7. Insertion – Insert the character "k". ("MathWor" to "MathWork")
8. Insertion – Insert the character "s". ("MathWork" to "MathWorks")

Compute the edit distance and specify the custom substitution cost function `caseInsensitiveSubstituteCost`, listed at the end of the example. The custom function `caseInsensitiveSubstituteCost` returns 0 if the two inputs are the same or differ only by case and returns 1 otherwise.

```
d = editDistance("MATLAB","MathWorks",'SubstituteCost',@caseInsensitiveSubstituteCost)
```

```
d = 6
```

This means the total cost for changing the first string to the second is 6. For example:

1. Substitution (cost 0) – Substitute the character "A" with "a". ("MATLAB" to "MaTLAB")
2. Substitution (cost 0) – Substitute the character "T" with "t". ("MaTLAB" to "MatLAB")
3. Substitution (cost 1) – Substitute the character "L" with "h". ("MatLAB" to "MathAB")
4. Substitution (cost 1) – Substitute the character "A" with "W". ("MathAB" to "MathWB")
5. Substitution (cost 1) – Substitute the character "B" with "o". ("MathWB" to "MathWo")
6. Insert (cost 1) – Insert the character "r". ("MathWo" to "MathWor")
7. Insert (cost 1) – Insert the character "k". ("MathWor" to "MathWork")
8. Insert (cost 1) – Insert the character "s". ("MathWork" to "MathWorks")

Custom Cost Function

The custom function `caseInsensitiveSubstituteCost` returns 0 if the two inputs are the same or differ only by case and returns 1 otherwise.

```
function cost = caseInsensitiveSubstituteCost(grapheme1,grapheme2)

if lower(grapheme1) == lower(grapheme2)
    cost = 0;
```

```
else
    cost = 1;
end

end
```

Input Arguments

str1 — Source string
string array | character vector | cell array of character vectors

Source string, specified as a string array, character vector, or a cell array of character vectors.

If `str1` contains multiple strings, then `str2` must be the same size as `str1` or scalar.

Data Types: `char` | `string` | `cell`

str2 — Target string
string array | character vector | cell array of character vectors

Target string, specified as a string array, character vector, or a cell array of character vectors.

If `str2` contains multiple strings, then `str1` must be the same size as `str2` or scalar.

Data Types: `char` | `string` | `cell`

document1 — Source document
`tokenizedDocument`

Source document, specified as a `tokenizedDocument` array.

If `document1` contains multiple documents, then `document2` must be the same size as `document1` or scalar.

document2 — Target document
`tokenizedDocument`

Target document, specified as a `tokenizedDocument` array.

If `document2` contains multiple documents, then `document1` must be the same size as `document2` or scalar.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `editDistance("MATALB","MATLAB",'SwapCost',1)` returns the edit distance between the strings `"MATALB"` and `"MATLAB"` and sets the cost to swap two adjacent graphemes to 1.

InsertCost — Cost to insert grapheme or token
1 (default) | nonnegative scalar | function handle

Cost to insert a grapheme or token, specified as the comma-separated pair consisting of `'InsertCost'` and a nonnegative scalar or a function handle.

If `'InsertCost'` is a function handle, then the function must accept a single input and return the cost of inserting the input to the source. For example:

- For string input to `editDistance`, the cost function must have the form `cost = func(grapheme)`, where the function returns the cost of inserting `grapheme` into `str1`.
- For document input to `editDistance`, the cost function must have the form `cost = func(token)`, where the function returns the cost of inserting `token` into `document1`.

Example: `'InsertCost',2`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

`DeleteCost` — Cost to delete grapheme or token
1 (default) | nonnegative scalar | function handle

Cost to delete grapheme or token, specified as the comma-separated pair consisting of `'DeleteCost'` and a nonnegative scalar or a function handle.

If `'DeleteCost'` is a function handle, then the function must accept a single input and return the cost of deleting the input from the source. For example:

- For string input to `editDistance`, the cost function must have the form `cost = func(grapheme)`, where the function returns the cost of deleting `grapheme` from `str1`.
- For document input to `editDistance`, the cost function must have the form `cost = func(token)`, where the function returns the cost of deleting `token` from `document1`.

Example: `'DeleteCost',2`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

`SubstituteCost` — Cost to substitute grapheme or token
1 (default) | nonnegative scalar | function handle

Cost to substitute a grapheme or token, specified as the comma-separated pair consisting of `'SubstituteCost'` and a nonnegative scalar or a function handle.

If `'SubstituteCost'` is a function handle, then the function must accept exactly two inputs and return the cost of substituting the first input with the second in the source. For example:

- For string input to `editDistance`, the cost function must have the form `cost = func(grapheme1,grapheme2)`, where the function returns the cost of substituting `grapheme1` with `grapheme2` in `str1`.
- For document input to `editDistance`, the cost function must have the form `cost = func(token1,token2)`, where the function returns the cost of substituting `token1` with `token2` in `document1`.

Example: `'SubstituteCost',2`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

`SwapCost` — Cost to swap two adjacent graphemes or tokens
`Inf` (default) | nonnegative scalar | function handle

Cost to swap two adjacent graphemes or tokens, specified as the comma-separated pair consisting of `'SwapCost'` and a nonnegative scalar or a function handle.

If `'SwapCost'` is a function handle, then the function must accept exactly two inputs and return the cost of swapping the first input with the second in the source. For example:

- For string input to `editDistance`, the cost function must have the form `cost = func(grapheme1,grapheme2)`, where the function returns the cost of swapping the adjacent graphemes `grapheme1` and `grapheme2` in `str1`.
- For document input to `editDistance`, the cost function must have the form `cost = func(token1,token2)`, where the function returns the cost of swapping the adjacent tokens `token1` and `token2` in `document1`.

Example: `'SwapCost',2`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

Output Arguments

d — Edit distance
nonnegative scalar

Edit distance, returned as a nonnegative scalar.

Algorithms

Edit Distance

The function, by default, uses the Levenshtein distance: the lowest number of insertions, deletions, and substitutions required to convert one string to another.

For other commonly used edit distances, use these options:

Distance	Description	Options
Levenshtein (default)	lowest number of insertions, deletions, and substitutions	Default
Damerau-Levenshtein	lowest number of insertions, deletions, substitutions, and swaps	`'SwapCost',1`
Hamming	lowest number of substitutions only	`'InsertCost',Inf,'DeleteCost',Inf`

See Also

`correctSpelling` | `editDistanceSearcher` | `knnsearch` | `rangesearch` | `splitGraphemes` | `tokenizedDocument`

Topics
"Correct Spelling in Documents"
"Create Extension Dictionary for Spelling Correction"
"Create Custom Spelling Correction Function Using Edit Distance Searchers"

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Analyze Text Data Using Topic Models”

Introduced in R2019a

editDistanceSearcher

Edit distance nearest neighbor searcher

Description

An edit distance searcher performs a nearest neighborhood search in a list of known strings, using edit distance.

Creation

Syntax

```
eds = editDistanceSearcher(vocabulary,maxDist)
eds = editDistanceSearcher(vocabulary,maxDist,Name,Value)
```

Description

`eds = editDistanceSearcher(vocabulary,maxDist)` creates an edit distance searcher and sets the `Vocabulary` and `MaximumDistance` properties. The returned object searches the words in `vocabulary` and with maximum edit distance `maxDist`.

`eds = editDistanceSearcher(vocabulary,maxDist,Name,Value)` specifies additional options using one or more name-value pair arguments.

Properties

`Vocabulary` — Words to compare against
string vector | character vector | cell array of character vectors

Words to compare against, specified as a string vector, character vector, or a cell array of character vectors.

Data Types: `char` | `string` | `cell`

`MaximumDistance` — Maximum edit distance
positive scalar

Maximum edit distance, specified as a positive scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`InsertCost` — Cost to insert grapheme
1 (default) | nonnegative scalar | function handle

Cost to insert grapheme, specified as a nonnegative scalar or a function handle.

If `InsertCost` is a function handle, then the function must accept a single input and return the cost of inserting the input to the source. The cost function must have the form `cost = func(grapheme)`, where the function returns the cost of inserting `grapheme` into the source string.

If you specify a custom cost function, then the searcher perform exhaustive searching. For large vocabularies, the functions `knnsearch` and `rangesearch` can take a long time to find matches.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

`DeleteCost` — Cost to delete grapheme
1 (default) | nonnegative scalar | function handle

Cost to delete grapheme, specified as a nonnegative scalar or a function handle.

If `DeleteCost` is a function handle, then the function must accept a single input and return the cost of deleting the input from the source. The cost function must have the form `cost = func(grapheme)`, where the function returns the cost of deleting `grapheme` from the source string.

If you specify a custom cost function, then the searcher perform exhaustive searching. For large vocabularies, the functions `knnsearch` and `rangesearch` can take a long time to find matches.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

`SubstituteCost` — Cost to substitute grapheme
1 (default) | nonnegative scalar | function handle

Cost to substitute grapheme, specified as a nonnegative scalar or a function handle.

If `SubstituteCost` is a function handle, then the function must accept exactly two inputs and return the cost of substituting the first input to the second in the source. The cost function must have the form `cost = func(grapheme1,grapheme2)`, where the function returns the cost of substituting `grapheme1` with `grapheme2` in the source.

If you specify a custom cost function, then the searcher perform exhaustive searching. For large vocabularies, the functions `knnsearch` and `rangesearch` can take a long time to find matches.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

`SwapCost` — Cost to swap adjacent graphemes
`Inf` (default) | nonnegative scalar | function handle

Cost to swap adjacent graphemes, specified as a nonnegative scalar or a function handle.

If `SwapCost` is a function handle, then the function must accept exactly two inputs and return the cost of swapping the first input with the second in the source. The cost function must have the form `cost = func(grapheme1,grapheme2)`, where the function returns the cost of swapping the adjacent graphemes `grapheme1` and `grapheme2` in the source.

If you specify a custom cost function, then the searcher perform exhaustive searching. For large vocabularies, the functions `knnsearch` and `rangesearch` can take a long time to find matches.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `function_handle`

Object Functions

rangesearch	Find nearest neighbors by edit distance range

knnsearch	Find nearest neighbors by edit distance

Examples

Create Edit Distance Searcher

Create an edit distance searcher with a maximum edit distance 3 from the words "MathWorks", "MATLAB", and "Analytics".

```
vocabulary = ["MathWorks" "MATLAB" "Analytics"];
eds = editDistanceSearcher(vocabulary,3)
```

```
eds = 
  editDistanceSearcher with properties:

         Vocabulary: ["MathWorks"    "MATLAB"    "Analytics"]
    MaximumDistance: 3
         InsertCost: 1
         DeleteCost: 1
     SubstituteCost: 1
           SwapCost: Inf
```

Create Damerau-Levenshtein Edit Distance Searcher

Create an edit distance searcher using the Damerau-Levenshtein edit distance. The Damerau-Levenshtein edit distance is the lowest number of insertions, deletions, substitutions, and swaps.

Create the edit distance searcher from the words "MathWorks", "MATLAB", and "Analytics" and specify a maximum distance of 3. To specify the Damerau-Levenshtein edit distance, set 'SwapCost' to 1.

```
vocabulary = ["MathWorks" "MATLAB" "Analytics"];
eds = editDistanceSearcher(vocabulary,3,'SwapCost',1)
```

```
eds = 
  editDistanceSearcher with properties:

         Vocabulary: ["MathWorks"    "MATLAB"    "Analytics"]
    MaximumDistance: 3
         InsertCost: 1
         DeleteCost: 1
     SubstituteCost: 1
           SwapCost: 1
```

Find Nearest Words

Create an edit distance searcher.

```
vocabulary = ["Text" "Analytics" "Toolbox"];
eds = editDistanceSearcher(vocabulary,2);
```

Find the nearest words to `"Test"` and `"Analysis"`.

```
words = ["Test" "Analysis"];
idx = knnsearch(eds,words)
```

```
idx = 2×1

     1
     2
```

Get the words from the vocabulary using the returned indices.

```
nearestWords = eds.Vocabulary(idx)
```

```
nearestWords = 1x2 string
    "Text"    "Analytics"
```

Find Nearest Neighbors in Range

Create an edit distance searcher and specify a maximum edit distance of 3.

```
vocabulary = ["MathWorks" "MATLAB" "Simulink" "text" "analytics" "analysis"];
maxDist = 3;
eds = editDistanceSearcher(vocabulary,maxDist);
```

Find the nearest words to `"test"`, `"analytic"`, and `"analyze"` with edit distance less than or equal to 1.

```
words = ["test" "analytic" "analyze"];
maxDist = 1;
idx = rangesearch(eds,words,maxDist)
```

```
idx=3×1 cell array
    {[       4]}
    {[       5]}
    {1x0 double}
```

For `"analyze"`, there are no words in the searcher within the specified range. For `"test"` and `"analytic"`, there is one result each. View the corresponding word for `"test"` using the returned index.

```
nearestWords = eds.Vocabulary(idx{2})
```

```
nearestWords = 
"analytics"
```

Find the nearest words to `"test"`, `"analytic"`, and `"analyze"` with edit distance less than or equal to 3 and their corresponding edit distances.

```
words = ["test" "analytic" "analyze"];
maxDist = 3;
[idx,d] = rangesearch(eds,words,maxDist)

idx=3×1 cell array
    {[  4]}
    {[5 6]}
    {[  6]}


d=3×1 cell array
    {[  1]}
    {[1 2]}
    {[  3]}
```

For both `"test"` and `"analyze"`, there is one word in the searcher within the specified range. For `"analytic"`, there are two results. View the corresponding words for `"analytic"` (the second word) using the returned indices and their edit distances.

```
i = 2;
nearestWords = eds.Vocabulary(idx{i})

nearestWords = 1x2 string
    "analytics"    "analysis"


d{i}

ans = 1×2

     1     2
```

Algorithms

Edit Distance

The function, by default, uses the Levenshtein distance: the lowest number of insertions, deletions, and substitutions required to convert one string to another.

For other commonly used edit distances, use these options:

Distance	Description	Options
Levenshtein (default)	lowest number of insertions, deletions, and substitutions	Default
Damerau-Levenshtein	lowest number of insertions, deletions, substitutions, and swaps	`'SwapCost',1`
Hamming	lowest number of substitutions only	`'InsertCost',Inf,'DeleteCost',Inf`

See Also

`correctSpelling` | `editDistance` | `knnsearch` | `rangesearch` | `splitGraphemes` | `tokenizedDocument`

Topics

"Correct Spelling in Documents"
"Create Extension Dictionary for Spelling Correction"
"Create Custom Spelling Correction Function Using Edit Distance Searchers"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"

Introduced in R2019a

encode

Encode documents as matrix of word or n-gram counts

Syntax

```
counts = encode(bag,documents)
counts = encode(bag,words)
counts = encode( ___ ,Name,Value)
```

Description

Use `encode` to encode an array of tokenized documents as a matrix of word or n-gram counts according to a bag-of-words or bag-of-n-grams model. To encode documents as vectors or word indices, use a `wordEncoding` object.

`counts = encode(bag,documents)` returns a matrix of frequency counts for `documents` based on the bag-of-words or bag-of-n-grams model `bag`.

`counts = encode(bag,words)` returns a matrix of frequency counts for a list of words.

`counts = encode( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Encode Documents as Word Count Matrix

Encode an array of documents as a matrix of word counts.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [2x7 double]
      Vocabulary: [1x7 string]
        NumWords: 7
    NumDocuments: 2
```

```
documents = tokenizedDocument([
    "a new sentence"
    "a second new sentence"])
```

```
documents = 
  2x1 tokenizedDocument:

    3 tokens: a new sentence
    4 tokens: a second new sentence
```

View the documents encoded as a matrix of word counts. The word "new" does not appear in `bag`, so it is not counted.

```
counts = encode(bag,documents);
full(counts)
```

```
ans = 2×7

     0     0     0     1     0     1     0
     0     0     0     1     0     1     1
```

The columns correspond to the vocabulary of the bag-of-words model.

```
bag.Vocabulary
```

```
ans = 1x7 string
    "an"    "example"    "of"    "a"    "short"    "sentence"    "second"
```

Encode Words as Word Count Vector

Encode an array of words as a vector of word counts.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [2x7 double]
      Vocabulary: [1x7 string]
        NumWords: 7
    NumDocuments: 2
```

```
words = ["another" "example" "of" "a" "short" "example" "sentence"];
counts = encode(bag,words)
```

```
counts = 
   (1,2)        2
   (1,3)        1
   (1,4)        1
   (1,5)        1
   (1,6)        1
```

Output Document Word Counts in Columns

Encode an array of documents as a matrix of word counts with documents in columns.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents)
```

```
bag =
  bagOfWords with properties:

          Counts: [2x7 double]
      Vocabulary: [1x7 string]
        NumWords: 7
    NumDocuments: 2
```

```
documents = tokenizedDocument([
    "a new sentence"
    "a second new sentence"])
```

```
documents =
  2x1 tokenizedDocument:

    3 tokens: a new sentence
    4 tokens: a second new sentence
```

View the documents encoded as a matrix of word counts with documents in columns. The word "new" does not appear in `bag`, so it is not counted.

```
counts = encode(bag,documents,'DocumentsIn','columns');
full(counts)
```

```
ans = 7×2

     0     0
     0     0
     0     0
     1     1
     0     0
     1     1
     0     1
```

Input Arguments

bag — Input bag-of-words or bag-of-n-grams model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object.

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is a string array or a cell array of character vectors, then it must be a row vector representing a single document, where each element is a word.

Tip To ensure that the documents are encoded correctly, you must preprocess the input documents using the same steps as the documents used to create the input model. For an example showing how to create a function to preprocess text data, see “Prepare Text Data for Analysis”.

`words` — Input words

string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DocumentsIn','rows'` specifies the orientation of the output documents as rows.

`DocumentsIn` — Orientation of output documents

`'rows'` (default) | `'columns'`

Orientation of output documents in the frequency count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Return a matrix of frequency counts with rows corresponding to documents.
- `'columns'` – Return a transposed matrix of frequency counts with columns corresponding to documents.

Data Types: `char`

`ForceCellOutput` — Indicator for forcing output to be returned as cell array

`false` (default) | `true`

Indicator for forcing output to be returned as cell array, specified as the comma separated pair consisting of `'ForceCellOutput'` and `true` or `false`.

Data Types: `logical`

Output Arguments

`counts` — Word or n-gram counts

sparse matrix | cell array of sparse matrices

Word or n-gram counts, returned as a sparse matrix of nonnegative integers or a cell array of sparse matrices.

If `bag` is a non-scalar array or `'ForceCellOutput'` is `true`, then the function returns the outputs as a cell array of sparse matrices. Each element in the cell array is matrix of word or n-gram counts of the corresponding element of `bag`.

See Also

`bagOfNgrams` | `bagOfWords` | `tfidf` | `tokenizedDocument`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2017b

erasePunctuation

Erase punctuation from text and documents

Syntax

```
newStr = erasePunctuation(str)
newDocuments = erasePunctuation(documents)
newDocuments = erasePunctuation(documents,'TokenTypes',types)
```

Description

`newStr = erasePunctuation(str)` erases punctuation and symbols from the elements of `str`. The function removes characters that belong to the Unicode punctuation or symbol classes.

`newDocuments = erasePunctuation(documents)` erases punctuation and symbols from `documents`. If a word is empty after removing punctuation and symbol characters, then the function removes it. For tokenized document input, the function erases punctuation from tokens with type `'punctuation'` and `'other'`. For example, the function does not erase punctuation and symbol characters from URLs and email addresses.

`newDocuments = erasePunctuation(documents,'TokenTypes',types)` erases punctuation and symbols from only the specified token types.

Examples

Erase Punctuation from Text

Erase the punctuation from the text in `str`.

```
str = "it's one and/or two.";
newStr = erasePunctuation(str)
```

```
newStr = 
"its one andor two"
```

To insert a space where the `"/"` symbol is, first use the `replace` function.

```
newStr = replace(str,"/"," ")
```

```
newStr = 
"it's one and or two."
```

```
newStr = erasePunctuation(newStr)
```

```
newStr = 
"its one and or two"
```

Erase Punctuation from Documents

Erase the punctuation from an array of documents.

```
documents = tokenizedDocument([ ...
    "An example of a short sentence."
    "Another example... with a URL: https://www.mathworks.com"])
```

```
documents = 
  2x1 tokenizedDocument:

     7 tokens: An example of a short sentence .
    10 tokens: Another example . . . with a URL : https://www.mathworks.com
```

```
newDocuments = erasePunctuation(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: An example of a short sentence
    6 tokens: Another example with a URL https://www.mathworks.com
```

Here, the function does not erase the punctuation symbols from the URL.

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

Example: `["An example of a short sentence."; "A second short sentence."]`

Data Types: `string` | `char` | `cell`

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`types` — Token types to erase punctuation from
`{'punctuation','other'}` (default) | string array | character vector | cell array of character vectors

Token types to erase punctuation from, specified as a character vector, string array, or a cell array of character vectors containing one or more token types (including custom token types).

The `tokenizedDocument` and `addTypeDetails` functions automatically detect the following token types:

- `'letters'` – string of letter characters only
- `'digits'` – string of digits only
- `'punctuation'` – string of punctuation and symbol characters only

- `'email-address'` – detected email address
- `'web-address'` – detected web address
- `'hashtag'` – detected hashtag (starts with `"#"` character followed by a letter)
- `'at-mention'` – detected at-mention (starts with `"@"` character)
- `'emoticon'` – detected emoticon
- `'emoji'` – detected emoji
- `'other'` – does not belong to the previous types and is not a custom type

To specify your own custom token types when tokenizing, use the `'CustomTokens'` or `'RegularExpressions'` options in `tokenizedDocument`. If you do not specify a type for a custom token, then the software sets the corresponding token type to `'custom'`.

Data Types: `string` | `char` | `cell`

Output Arguments

`newStr` — Output text
string array | character vector | cell array of character vectors

Output text, returned as a string array, a character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

More About

Unicode Character Categories

Each Unicode character is assigned a category. The following table summarizes the Unicode punctuation and symbol categories and provides an example character from each category:

Category	Category Code	Number of Characters	Example Character
Punctuation, Connector	[Pc]	10	_
Punctuation, Dash	[Pd]	24	-
Punctuation, Close	[Pe]	73	)
Punctuation, Final quote	[Pf]	10	”
Punctuation, Initial quote	[Pi]	12	“
Punctuation, Other	[Po]	566	!
Punctuation, Open	[Ps]	75	(
Symbol, Currency	[Sc]	54	$
Symbol, Modifier	[Sk]	121	^

Category	Category Code	Number of Characters	Example Character
Symbol, Math	[Sm]	948	+
Symbol, Other	[So]	5855	¦

For more information, see [1].

Tips

- For string input, `erasePunctuation` removes punctuation characters from URLs and HTML tags. This behavior can prevent the functions `eraseTags`, `eraseURLs`, and `decodeHTMLEntities` from working as expected. If you want to use these functions to preprocess your text, then use these functions before using `erasePunctuation`.

Compatibility Considerations

`erasePunctuation` skips complex tokens
Behavior changed in R2018b

Starting in R2018b, for `tokenizedDocument` input, `erasePunctuation`, by default, erases punctuation and symbol characters from tokens with type `'punctuation'` or `'other'` only. This behavior prevents the function from affecting complex tokens such as URLs and email-addresses.

In previous versions, `erasePunctuation` erases punctuation characters from all tokens. To reproduce the behavior, use the `'TokenTypes'` name-value pair.

References

[1] *Unicode Character Categories*. https://www.fileformat.info/info/unicode/category/index.htm

See Also

`decodeHTMLEntities` | `eraseTags` | `eraseURLs` | `lower` | `tokenizedDocument` | `upper`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

eraseTags

Erase HTML and XML tags from text

Syntax

```
newStr = eraseTags(str)
```

Description

`newStr = eraseTags(str)` erases HTML and XML comments and tags from the elements of `str`.

The function erases comments and tags with tag name `a`, `abbr`, `acronym`, `b`, `bdi`, `bdo`, `big`, `code`, `del`, `dfn`, `em`, `font`, `i`, `ins`, `kbd`, `mark`, `rp`, `rt`, `ruby`, `s`, `small`, `span`, `strike`, `strong` `sub`, `sup`, `tt`, `u`, `var` and `wbr`, and replaces all other tags with a space.

The function does not remove HTML and XML elements (the tags as well anything between start and end tags). For example, `eraseTags("x<a>y</a>")` returns the string `"xy"`. It only removes the tags `<a>` and `</a>`, and does not remove the element `<a>y</a>`.

Examples

Erase HTML and XML Tags and Comments

Erase the tags from some HTML code. The function replaces the `<br>` tag with a space.

```
htmlCode = "one.<br>two";
newStr = eraseTags(htmlCode)
```

```
newStr = 
"one. two"
```

Erase the tags from some XML code. The function removes the `<sub>` tags and does not replace them with a space.

```
xmlCode = "H<sub>2</sub>O";
newStr = eraseTags(xmlCode)
```

```
newStr = 
"H2O"
```

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

Example: `["An example of a short sentence."; "A second short sentence."]`

Data Types: `string` | `char` | `cell`

Output Arguments

`newStr` — Output text
string array | character vector | cell array of character vectors

Output text, returned as a string array, a character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

See Also

`decodeHTMLEntities` | `erasePunctuation` | `eraseURLs` | `lower` | `tokenizedDocument` | `upper`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

eraseURLs

Erase HTTP and HTTPS URLs from text

Syntax

```
newStr = eraseURLs(str)
```

Description

`newStr = eraseURLs(str)` erases HTTP and HTTPS URLs from the elements of `str`.

Examples

Erase URL from Text

Erase the URL from the text in `str`.

```
str = "For more information, see https://www.mathworks.com";
newStr = eraseURLs(str)
```

```
newStr = 
"For more information, see "
```

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

Example: `["An example of a short sentence."; "A second short sentence."]`

Data Types: `string` | `char` | `cell`

Output Arguments

`newStr` — Output text
string array | character vector | cell array of character vectors

Output text, returned as a string array, a character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

See Also

`decodeHTMLEntities` | `erasePunctuation` | `eraseTags` | `lower` | `tokenizedDocument` | `upper`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

extractFileText

Read text from PDF, Microsoft Word, HTML, and plain text files

Syntax

```
str = extractFileText(filename)
str = extractFileText(filename,Name,Value)
```

Description

`str = extractFileText(filename)` reads the text data from a file as a string.

`str = extractFileText(filename,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Extract Text Data from Text File

Extract the text from `sonnets.txt` using `extractFileText`. The file `sonnets.txt` contains Shakespeare's sonnets in plain text.

```
str = extractFileText("sonnets.txt");
```

View the first sonnet.

```
i = strfind(str,"I");
ii = strfind(str,"II");
start = i(1);
fin = ii(1);
extractBetween(str,start,fin-1)
```

```
ans = 
    "I
    
       From fairest creatures we desire increase,
       That thereby beauty's rose might never die,
       But as the riper should by time decease,
       His tender heir might bear his memory:
       But thou, contracted to thine own bright eyes,
       Feed'st thy light's flame with self-substantial fuel,
       Making a famine where abundance lies,
       Thy self thy foe, to thy sweet self too cruel:
       Thou that art now the world's fresh ornament,
       And only herald to the gaudy spring,
       Within thine own bud buriest thy content,
       And tender churl mak'st waste in niggarding:
         Pity the world, or else this glutton be,
         To eat the world's due, by the grave and thee.
    
       "
```

Extract Text Data from PDF

Extract the text from `exampleSonnets.pdf` using `extractFileText`. The file `exampleSonnets.pdf` contains Shakespeare's sonnets in a PDF file.

```
str = extractFileText("exampleSonnets.pdf");
```

View the second sonnet.

```
ii = strfind(str,"II");
iii = strfind(str,"III");
start = ii(1);
fin = iii(1);
extractBetween(str,start,fin-1)
```

```
ans = 
    "II
    
       When forty winters shall besiege thy brow,
       And dig deep trenches in thy beauty's field,
       Thy youth's proud livery so gazed on now,
       Will be a tatter'd weed of small worth held:
       Then being asked, where all thy beauty lies,
       Where all the treasure of thy lusty days;
       To say, within thine own deep sunken eyes,
       Were an all-eating shame, and thriftless praise.
       How much more praise deserv'd thy beauty's use,
       If thou couldst answer 'This fair child of mine
       Shall sum my count, and make my old excuse,'
       Proving his beauty by succession thine!
         This were to be new made when thou art old,
         And see thy blood warm when thou feel'st it cold.
    
       "
```

Extract the text from pages 3, 5, and 7 of the PDF file.

```
pages = [3 5 7];
str = extractFileText("exampleSonnets.pdf", ...
    'Pages',pages);
```

View the 10th sonnet.

```
x = strfind(str,"X");
xi = strfind(str,"XI");
start = x(1);
fin = xi(1);
extractBetween(str,start,fin-1)
```

```
ans = 
    "X
    
       Is it for fear to wet a widow's eye,
       That thou consum'st thy self in single life?
       Ah! if thou issueless shalt hap to die,
       The world will wail thee like a makeless wife;
       The world will be thy widow and still weep
       That thou no form of thee hast left behind,
       When every private widow well may keep
       By children's eyes, her husband's shape in mind:
       Look! what an unthrift in the world doth spend
       Shifts but his place, for still the world enjoys it;
       But beauty's waste hath in the world an end,
       And kept unused the user so destroys it.
         No love toward others in that bosom sits
         That on himself such murd'rous shame commits.

       X

       For shame! deny that thou bear'st love to any,
       Who for thy self art so unprovident.
       Grant, if thou wilt, thou art belov'd of many,
       But that thou none lov'st is most evident:
       For thou art so possess'd with murderous hate,
       That 'gainst thy self thou stick'st not to conspire,
       Seeking that beauteous roof to ruinate
       Which to repair should be thy chief desire.



       "
```

Import Text from Multiple Files Using a File Datastore

If your text data is contained in multiple files in a folder, then you can import the text data into MATLAB using a file datastore.

Create a file datastore for the example sonnet text files. The examples sonnets have file names "`exampleSonnetN.txt`", where `N` is the number of the sonnet. Specify the read function to be `extractFileText`.

```
readFcn = @extractFileText;
fds = fileDatastore('exampleSonnet*.txt','ReadFcn',readFcn)
```

```
fds = 
  FileDatastore with properties:

                       Files: {
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonnet
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonnet
                              ' ...\ib8F3FCD\26\tp5980089c\textanalytics-ex73762432\exampleSonnet
                               ... and 1 more
                              }
                     Folders: {
                              ' ...\Bdoc21a_1606923_5032\ib8F3FCD\26\tp5980089c\textanalytics-ex
                              }
                 UniformRead: 0
                    ReadMode: 'file'
                   BlockSize: Inf
                  PreviewFcn: @extractFileText
      SupportedOutputFormats: [1x16 string]
```

```
                     ReadFcn: @extractFileText
    AlternateFileSystemRoots: {}
```

Create an empty bag-of-words model.

```
bag = bagOfWords
```

```
bag = 
  bagOfWords with properties:

          Counts: []
      Vocabulary: [1x0 string]
        NumWords: 0
    NumDocuments: 0
```

Loop over the files in the datastore and read each file. Tokenize the text in each file and add the document to `bag`.

```
while hasdata(fds)
    str = read(fds);
    document = tokenizedDocument(str);
    bag = addDocument(bag,document);
end
```

View the updated bag-of-words model.

```
bag
```

```
bag = 
  bagOfWords with properties:

          Counts: [4x276 double]
      Vocabulary: [1x276 string]
        NumWords: 276
    NumDocuments: 4
```

Extract Text from HTML

To extract text data directly from HTML code, use `extractHTMLText` and specify the HTML code as a string.

```
code = "<html><body><h1>THE SONNETS</h1><p>by William Shakespeare</p></body></html>";
str = extractHTMLText(code)
```

```
str = 
    "THE SONNETS

     by William Shakespeare"
```

Input Arguments

`filename` — Name of file
string scalar | character vector

Name of the file, specified as a string scalar or character vector.

Data Types: `string` | `char`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Pages',[1 3 5]` specifies to read pages 1, 3, and 5 from a PDF file.

`Encoding` — Character encoding
`'auto'` (default) | `'UTF-8'` | `'ISO-8859-1'` | `'windows-1251'` | `'windows-1252'` | ...

Character encoding to use, specified as the comma-separated pair consisting of `'Encoding'` and a character vector or a string scalar. The character vector or string scalar must contain a standard character encoding scheme name such as the following.

'Big5'	'ISO-8859-1'	'windows-874'
'Big5-HKSCS'	'ISO-8859-2'	'windows-949'
'CP949'	'ISO-8859-3'	'windows-1250'
'EUC-KR'	'ISO-8859-4'	'windows-1251'
'EUC-JP'	'ISO-8859-5'	'windows-1252'
'EUC-TW'	'ISO-8859-6'	'windows-1253'
'GB18030'	'ISO-8859-7'	'windows-1254'
'GB2312'	'ISO-8859-8'	'windows-1255'
'GBK'	'ISO-8859-9'	'windows-1256'
'IBM866'	'ISO-8859-11'	'windows-1257'
'KOI8-R'	'ISO-8859-13'	'windows-1258'
'KOI8-U'	'ISO-8859-15'	'US-ASCII'
	'Macintosh'	'UTF-8'
	'Shift_JIS'	

If you do not specify an encoding scheme, then the function performs heuristic auto-detection for the encoding to use. The heuristics depend on your locale. If these heuristics fail, then you must specify one explicitly.

This option only applies when the input is a plain text file.

Data Types: `char` | `string`

`ExtractionMethod` — Extraction method
`'tree'` (default) | `'article'` | `'all-text'`

Extraction method, specified as the comma-separated pair consisting of `'ExtractionMethod'` and one of the following:

Option	Description
`'tree'`	Analyze the DOM tree and text contents, then extract a block of paragraphs.
`'article'`	Detect article text and extract a block of paragraphs.
`'all-text'`	Extract all text in the HTML body, except for scripts and CSS styles.

This option supports HTML file input only.

`Password` — Password to open PDF file
character vector | string scalar

Password to open PDF file, specified as the comma-separated pair consisting of `'Password'` and a character vector or a string scalar. This option only applies if the input file is a PDF.

Example: `'Password','skroWhtaM'`

Data Types: `char` | `string`

`Pages` — Pages to read from PDF file
vector of positive integers

Pages to read from PDF file, specified as the comma-separated pair consisting of `'Pages'` and a vector of positive integers. This option only applies if the input file is a PDF file. The function, by default, reads all pages from the PDF file.

Example: `'Pages',[1 3 5]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Tips

- To read text directly from HTML code, use `extractHTMLText`.

Compatibility Considerations

`extractFileText` will no longer support extracting text from Microsoft Word 97–2003 binary DOC files
Not recommended starting in R2020b

Support for extracting text from Microsoft® Word 97–2003 binary DOC files using the `extractFileText` function will be removed in a future release. Microsoft Word DOCX files will continue to be supported.

To extract text data from Microsoft Word 97–2003 binary DOC files, first save the file as a PDF, Microsoft Word DOCX, HTML, or plain text file, then use the `extractFileText` function.

See Also

`extractHTMLText` | `readPDFFormData` | `tokenizedDocument` | `writeTextDocument`

Topics
"Extract Text Data from Files"

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

extractHTMLText

Extract text from HTML

Syntax

```
str = extractHTMLText(code)
str = extractHTMLText(tree)
str = extractHTMLText( ___ ,'ExtractionMethod',ex)
```

Description

`str = extractHTMLText(code)` parses the HTML code in `code` and extracts the text.

`str = extractHTMLText(tree)` extracts the text from an HTML tree.

`str = extractHTMLText( ___ ,'ExtractionMethod',ex)` also specifies the extraction method to use.

Examples

Extract Text from HTML

To extract text data directly from HTML code, use `extractHTMLText` and specify the HTML code as a string.

```
code = "<html><body><h1>THE SONNETS</h1><p>by William Shakespeare</p></body></html>";
str = extractHTMLText(code)
```

```
str = 
    "THE SONNETS

     by William Shakespeare"
```

Extract Text from Website

To extract the text data from a web page, first use the `webread` function to read the HTML code. Then use the `extractHTMLText` function on the returned code.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
str = extractHTMLText(code)
```

```
str = 
    'Text Analytics Toolbox™ provides algorithms and visualizations for preprocessing, analyzing

     Text Analytics Toolbox includes tools for processing raw text from sources such as equipmen

     Using machine learning techniques such as LSA, LDA, and word embeddings, you can find cluste
```

Find Elements in HTML Tree

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using the `webread` function.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

Find all the hyperlinks in the HTML tree using `findElement`. The hyperlinks are nodes with element name `"A"`.

```
selector = "A";
subtrees = findElement(tree,selector);
```

View the first few subtrees.

```
subtrees(1:10)
```

```
ans = 
  10×1 htmlTree:

    <A class="skip_link sr-only" href="#content_container">Skip to content</A>
    <A href="https://www.mathworks.com?s_tid=gn_logo" class="svg_link navbar-brand"><IMG src="/i
    <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Products</A>
    <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solutions</A>
    <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Academia</A>
    <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Support</A>
    <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Community</A>
    <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">Events</A>
    <A href="https://www.mathworks.com/products/get-matlab.html?s_tid=gn_getml">Get MATLAB</A>
    <A href="https://www.mathworks.com?s_tid=gn_logo" class="svg_link pull-left"><IMG src="/image
```

Extract the text from the subtrees using `extractHTMLText`. The result contains the link text from each link on the page.

```
str = extractHTMLText(subtrees);
str(1:10)
```

```
ans = 10×1 string
    "Skip to content"
    ""
    "Products"
    "Solutions"
    "Academia"
    "Support"
    "Community"
    "Events"
    "Get MATLAB"
    ""
```

Input Arguments

code — HTML code
string array | character vector | cell array of character vectors

HTML code, specified as a string array, a character vector, or a cell array of character vectors.

Tip

- To read HTML code from a web page, use `webread`.
- To extract text from an HTML file, use `extractFileText`.

Example: `"<a href='https://www.mathworks.com'>MathWorks</a>"`

Data Types: `char` | `string` | `cell`

`tree` — HTML tree
`htmlTree` array

HTML tree, specified as an `htmlTree` array.

ex — Extraction method
`'tree'` (default) | `'article'` | `'all-text'`

Extraction method, specified as one of the following:

Option	Description
`'tree'`	Analyze the DOM tree and text contents, then extract a block of paragraphs.
`'article'`	Detect article text and extract a block of paragraphs.
`'all-text'`	Extract all text in the HTML body, except for scripts and CSS styles.

See Also
`extractFileText` | `htmlTree` | `readPDFFormData` | `tokenizedDocument` | `webread` | `writeTextDocument`

Topics
"Parse HTML and Extract Text Content"
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018a

extractSummary

Extract summary from documents

Syntax

```
summary = extractSummary(documents)
[summary,scores] = extractSummary(documents)
[summary,scores] = extractSummary(documents,Name,Value)
```

Description

`summary = extractSummary(documents)` chooses a subset of the input documents to serve as a summary, and returns them as a `tokenizedDocument` array.

`[summary,scores] = extractSummary(documents)` also returns the importance scores used for selecting the summary documents. In this case, `scores(i)` represents the score for `summary(i)`.

`[summary,scores] = extractSummary(documents,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Summarize Documents

Create an array of tokenized documents.

```
str = [
    "The quick brown fox jumped over the lazy dog."
    "The fox jumped over the dog."
    "The lazy dog saw a fox jumping."
    "There seem to be animals jumping other animals."
    "There are quick animals and lazy animals"];
documents = tokenizedDocument(str);
```

Extract a summary of the documents using the `extractSummary` function. The function, by default, chooses 1/10 of the input documents, rounding up.

```
summary = extractSummary(documents)
```

```
summary = 
  tokenizedDocument:

   10 tokens: The quick brown fox jumped over the lazy dog .
```

To specify a larger summary, use the `'SummarySize'` option. Extract a three-document summary.

```
summary = extractSummary(documents,'SummarySize',3)
```

```
summary = 
  3x1 tokenizedDocument:
    10 tokens: The quick brown fox jumped over the lazy dog .
     7 tokens: The fox jumped over the dog .
     9 tokens: There seem to be animals jumping other animals .
```

Evaluate Document Importance

Create an array of tokenized documents.

```
str = [
    "The quick brown fox jumped over the lazy dog."
    "The fox jumped over the dog."
    "The lazy dog saw a fox jumping."
    "There seem to be animals jumping over other animals."
    "There are quick animals and lazy animals"];
documents = tokenizedDocument(str);
```

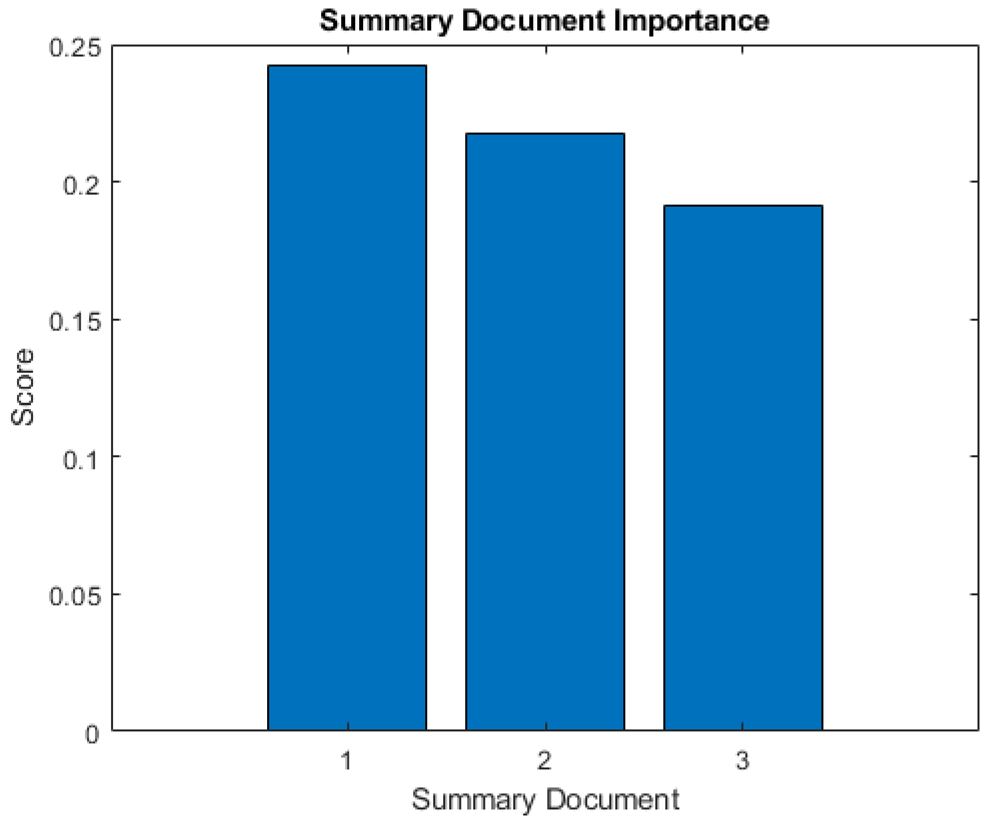
Extract a three-document summary. The second output `scores` contains the summary document importance scores.

```
[summary,scores] = extractSummary(documents,'SummarySize',3)
```

```
summary = 
  3x1 tokenizedDocument:

    10 tokens: The quick brown fox jumped over the lazy dog .
    10 tokens: There seem to be animals jumping over other animals .
     7 tokens: The fox jumped over the dog .


scores = 3×1

    0.2426
    0.2174
    0.1911
```

Visualize the scores in a bar chart.

```
figure
bar(scores)
xlabel("Summary Document")
ylabel("Score")
title("Summary Document Importance")
```



Sentence Level Summarization

To summarize a single document, split the document into an array of sentences, and use the `extractSummary` function.

Create a string scalar containing the document.

```
str = ...
    "There is a quick fox. The fox is brown. There is a dog which " + ...
    "is lazy. The dog is very lazy. The fox jumped over the dog. " + ...
    "The quick brown fox jumped over the lazy dog.";
```

Split the string into sentences using the `splitSentences` function.

```
str = splitSentences(str)
```

```
str = 6x1 string
    "There is a quick fox."
    "The fox is brown."
    "There is a dog which is lazy."
    "The dog is very lazy."
    "The fox jumped over the dog."
    "The quick brown fox jumped over the lazy dog."
```

Create a tokenized document array containing the sentences.

```
documents = tokenizedDocument(str)
```

```
documents = 
  6x1 tokenizedDocument:

     6 tokens: There is a quick fox .
     5 tokens: The fox is brown .
     8 tokens: There is a dog which is lazy .
     6 tokens: The dog is very lazy .
     7 tokens: The fox jumped over the dog .
    10 tokens: The quick brown fox jumped over the lazy dog .
```

Extract a summary from the sentences using the `extractSummary` function. To return a summary withthree documents, set the `'SummarySize'` option to 3.To ensure the summary documents appear in the same order as the input documents, set the `'OrderBy'` option to `'position'`.

```
summary = extractSummary(documents,'SummarySize',3,'OrderBy','position')
```

```
summary = 
  3x1 tokenizedDocument:

     6 tokens: There is a quick fox .
     7 tokens: The fox jumped over the dog .
    10 tokens: The quick brown fox jumped over the lazy dog .
```

To reconstruct the sentences into a single document, convert the documents to string using the `joinWords` function and join the sentences using the `join` function.

```
sentences = joinWords(summary);
summaryStr = join(sentences)
```

```
summaryStr = 
"There is a quick fox . The fox jumped over the dog . The quick brown fox jumped over the lazy d
```

To remove the surrounding punctuation characters, use the `replace` function.

```
punctuationRight = ["." "," "’" ")" ":" "?" "!"];
summaryStr = replace(summaryStr," " + punctuationRight,punctuationRight);

punctuationLeft = ["(" "‘"];
summaryStr = replace(summaryStr,punctuationLeft + " ",punctuationLeft)
```

```
summaryStr = 
"There is a quick fox. The fox jumped over the dog. The quick brown fox jumped over the lazy dog
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `extractSummary(documents,'ScoringMethod','lexrank')` extracts a summary from `documents` and sets the scoring method option to `'lexrank'`.

`ScoringMethod` — Scoring method

`'textrank'` (default) | `'lexrank'` | `'mmr'`

Scoring method used for extractive summarization, specified as the comma-separated pair consisting of `'ScoringMethod'` and one of the following:

- `'textrank'` – Use the TextRank algorithm.
- `'lexrank'` – Use the LexRank algorithm.
- `'mmr'` – Use the MMR algorithm.

`Query` — Query document for MMR scoring

`tokenizedDocument` scalar | string array | cell array of character vectors

Query document for MMR scoring, specified as the comma-separated pair consisting of `'Query'` and a `tokenizedDocument` scalar, a string array of words, or a cell array of character vectors. If `'Query'` not a `tokenizedDocument` scalar, then it must be a row vector representing a single document, where each element is a word.

This option only has an effect when `'ScoringMethod'` is `'mmr'`.

`SummarySize` — Size of summary

0.1 (default) | scalar in the range (0,1) | positive integer | `Inf`

Size of summary, specified as the comma-separated pair consisting of `'SummarySize'` and one of the following:

- Scalar in the range (0,1) – Extract the specified proportion of input documents, rounding up. In this case, the number of summary documents `ceil(SummarySize*numDocuments)`, where `numDocuments` is the number of input documents.
- Positive integer – Extract a summary with the specified number of documents. If `SummarySize` is greater than or equal to the number of input documents, then the function returns the input documents sorted according to the `'OrderBy'` option.

  `Inf` – Return the input documents sorted according to the `'OrderBy'` option.

Data Types: `double`

`OrderBy` — Order of documents in summary

`'score'` (default) | `'position'`

Order of documents in summary, specified as the comma-separated pair consisting of `'OrderBy'` and one of the following:

- `'score'` – Order documents by their score according to the `'ScoringMethod'` option.
- `'position'` – Maintain the document order from the input.

Output Arguments

`summary` — Extracted summary
`tokenizedDocument` array

Extracted summary, returned as a `tokenizedDocument` array. The summary is a subset of `documents`, and is sorted according to the `'OrderBy'` option.

`scores` — Summary document scores
vector

Summary document scores, returned as a vector, where `scores(i)` is the score of the `j`th summary document according to the `'ScoringMethod'` option. The scores are sorted according to the `'OrderBy'` option.

See Also

`bleuEvaluationScore` | `bm25Similarity` | `cosineSimilarity` | `lexrankScores` | `mmrScores` | `rakeKeywords` | `rougeEvaluationScore` | `textrankKeywords` | `textrankScores` | `tokenizedDocument`

Topics
"Extract Keywords from Text Data Using TextRank"
"Extract Keywords from Text Data Using RAKE"
"Sequence-to-Sequence Translation Using Attention"

Introduced in R2020a

fastTextWordEmbedding

Pretrained fastText word embedding

Syntax

`emb = fastTextWordEmbedding`

Description

`emb = fastTextWordEmbedding` returns a 300-dimensional pretrained word embedding for 1 million English words.

This function requires the Text Analytics Toolbox Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, the function provides a download link.

Examples

Download fastText Support Package

Download and install the Text Analytics Toolbox Model *for fastText English 16 Billion Token Word Embedding* support package.

Type `fastTextWordEmbedding` at the command line.

```
fastTextWordEmbedding
```

If the Text Analytics Toolbox Model *for fastText English 16 Billion Token Word Embedding* support package is not installed, then the function provides a link to the required support package in the Add-On Explorer. To install the support package, click the link, and then click **Install**. Check that the installation is successful by typing `emb = fastTextWordEmbedding` at the command line.

```
emb = fastTextWordEmbedding
```

```
emb = 

  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

If the required support package is installed, then the function returns a `wordEmbedding` object.

Map Words to Vectors and Back

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

Map the words "Italy", "Rome", and "Paris" to vectors using `word2vec`.

```
italy = word2vec(emb,"Italy");
rome = word2vec(emb,"Rome");
paris = word2vec(emb,"Paris");
```

Map the vector `italy - rome + paris` to a word using `vec2word`.

```
word = vec2word(emb,italy - rome + paris)
```

```
word = 
"France"
```

Convert Documents to Sequences of Word Vectors

Convert an array of tokenized documents to sequences of word vectors using a pretrained word embedding.

Load a pretrained word embedding using the `fastTextWordEmbedding` function. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding;
```

Load the factory reports data and create a `tokenizedDocument` array.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
textData = data.Description;
documents = tokenizedDocument(textData);
```

Convert the documents to sequences of word vectors using `doc2sequence`. The `doc2sequence` function, by default, left-pads the sequences to have the same length. When converting large collections of documents using a high-dimensional word embedding, padding can require large amounts of memory. To prevent the function from padding the data, set the `'PaddingDirection'` option to `'none'`. Alternatively, you can control the amount of padding using the `'Length'` option.

```
sequences = doc2sequence(emb,documents,'PaddingDirection','none');
```

View the sizes of the first 10 sequences. Each sequence is *D*-by-*S* matrix, where *D* is the embedding dimension, and *S* is the number of word vectors in the sequence.

```
sequences(1:10)
```

```
ans=10×1 cell array
    {300×10 single}
    {300×11 single}
    {300×11 single}
    {300×6  single}
    {300×5  single}
    {300×10 single}
    {300×8  single}
    {300×9  single}
    {300×7  single}
    {300×13 single}
```

Output Arguments

emb — Pretrained word embedding
`wordEmbedding` object

Pretrained word embedding, returned as a `wordEmbedding` object.

See Also

`doc2sequence` | `isVocabularyWord` | `readWordEmbedding` | `tokenizedDocument` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding`

Topics
“Train a Sentiment Classifier”
“Classify Text Data Using Deep Learning”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2018a

findElement

Find elements in HTML tree

Syntax

```
subtrees = findElement(tree,selector)
```

Description

`subtrees = findElement(tree,selector)` returns the elements in `tree` matching the CSS selector.

Examples

Find Elements in HTML Tree

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using the `webread` function.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

Find all the hyperlinks in the HTML tree using `findElement`. The hyperlinks are nodes with element name `"A"`.

```
selector = "A";
subtrees = findElement(tree,selector);
```

View the first few subtrees.

```
subtrees(1:10)
```

```
ans =
  10×1 htmlTree:

    <A class="skip_link sr-only" href="#content_container">Skip to content</A>
    <A href="https://www.mathworks.com?s_tid=gn_logo" class="svg_link navbar-brand"><IMG src="/i
    <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Products</A>
    <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solutions</A>
    <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Academia</A>
    <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Support</A>
    <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Community</A>
    <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">Events</A>
    <A href="https://www.mathworks.com/products/get-matlab.html?s_tid=gn_getml">Get MATLAB</A>
    <A href="https://www.mathworks.com?s_tid=gn_logo" class="svg_link pull-left"><IMG src="/imag
```

Extract the text from the subtrees using `extractHTMLText`. The result contains the link text from each link on the page.

```
str = extractHTMLText(subtrees);
str(1:10)
```

```
ans = 10×1 string
    "Skip to content"
    ""
    "Products"
    "Solutions"
    "Academia"
    "Support"
    "Community"
    "Events"
    "Get MATLAB"
    ""
```

Input Arguments

`tree` — HTML tree
scalar `htmlTree` object

HTML tree, specified as a scalar `htmlTree` object.

`selector` — CSS selector
string scalar | character vector

CSS selector, specified as a string scalar or a character vector. For more information, see “CSS Selectors” on page 1-154.

Output Arguments

`subtrees` — Matching HTML subtrees
`htmlTree` array

Matching HTML subtrees, returned as an `htmlTree` array.

More About

HTML Elements

A typical HTML element contains the following components:

- Element name – Name of the HTML tag. The element name corresponds to the `Name` property of the HTML tree.
- Attributes – Additional information about the tag. HTML attributes have the form *`name`*`="`*`value`*`"`, where *`name`* and *`value`* denote the attribute name and value respectively. The attributes appear inside the opening HTML tag. To get the attribute values from an HTML tree, use `getAttribute`.
- Content – Element content. The content appears between opening and closing HTML tags. The content can be text data or nested HTML elements. To extract the text from an `htmlTree` object,

use `extractHTMLText`. To get the nested HTML elements of an `htmlTree` object, use the `Children` property.

For example, the HTML element `<a href="https://www.mathworks.com">Home</a>` comprises the following components:

Component		Value	Description
Element name		`a`	Element is a hyperlink
Attribute	Attribute name	`href`	Hyperlink reference
	Attribute value	`"https://www.mathworks.com"`	Hyperlink reference value
Content		`Home`	Text to display

CSS Selectors

CSS selectors specify patterns to match elements in a tree.

This table shows some examples showing how to extract different HTML elements from an HTML tree:

Task	CSS Selector	Example
Find all paragraph (`<p>`) elements.	`"p"`	`findElement(tree,"p")`
Find all paragraph (`<p>`) and list item (`<li>`) elements.	`"p,li"`	`findElement(tree,"p,li")`
Find all paragraph (`<p>`) elements that are inside table (`<table>`) elements.	`"table p"`	`findElement(tree,"table p")`
Find all hyperlink (`<a>`) elements with hyperlink reference attribute (`href`) values ending with `".pdf"`.	`"a[href$="".pdf""]"`	`findElement(tree,"a[href$="".pdf""]")`
Find all paragraph (`<p>`) elements that are the first child of their parent.	`"p:first-child"`	`findElement(tr,"p:first-child")`
Find all paragraph (`<p>`) elements that are the first paragraph element of their parent.	`"p:first-of-type"`	`findElement(tr,"p:first-of-type")`
Find all emphasis (`<em>`) elements where the parent is a paragraph (`<p>`) element.	`"p > em"`	`findElement(tr,"p > em")`
Find all paragraph (`<p>`) elements appearing immediately after a heading 1 (`<h1>`) element	`"h1 + p"`	`findElement(tr,"h1 + p")`
Find all empty elements.	`":empty"`	`findElement(tr,":empty")`

Task	CSS Selector	Example
Find all nonempty label (`<label>`) elements.	`"label:not(:empty)"`	`findElement(tr,"label:not(:empty)")`

The `findElement` function supports all of CSS level 3, except for the selectors `":lang"`, `":checked"`, `":link"`, `":active"`, `":hover"`, `":focus"`, `":target"`, `":enabled"`, and `":disabled"`.

For more information about CSS selectors, see [1].

References

[1] *CSS Selector Reference*. https://www.w3schools.com/cssref/css_selectors.asp

See Also

`extractFileText` | `extractHTMLText` | `getAttribute` | `htmlTree` | `ismissing` | `readPDFFormData` | `tokenizedDocument`

Topics

"Parse HTML and Extract Text Content"
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018b

fitlda

Fit latent Dirichlet allocation (LDA) model

Syntax

```
mdl = fitlda(bag,numTopics)
mdl = fitlda(counts,numTopics)
mdl = fitlda(___,Name,Value)
```

Description

A latent Dirichlet allocation (LDA) model is a topic model which discovers underlying topics in a collection of documents and infers word probabilities in topics. If the model was fit using a bag-of-n-grams model, then the software treats the n-grams as individual words.

`mdl = fitlda(bag,numTopics)` fits an LDA model with `numTopics` topics to the bag-of-words or bag-of-n-grams model `bag`.

`mdl = fitlda(counts,numTopics)` fits an LDA model to the documents represented by a matrix of frequency counts.

`mdl = fitlda(___,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Fit LDA Model

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
```

```
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

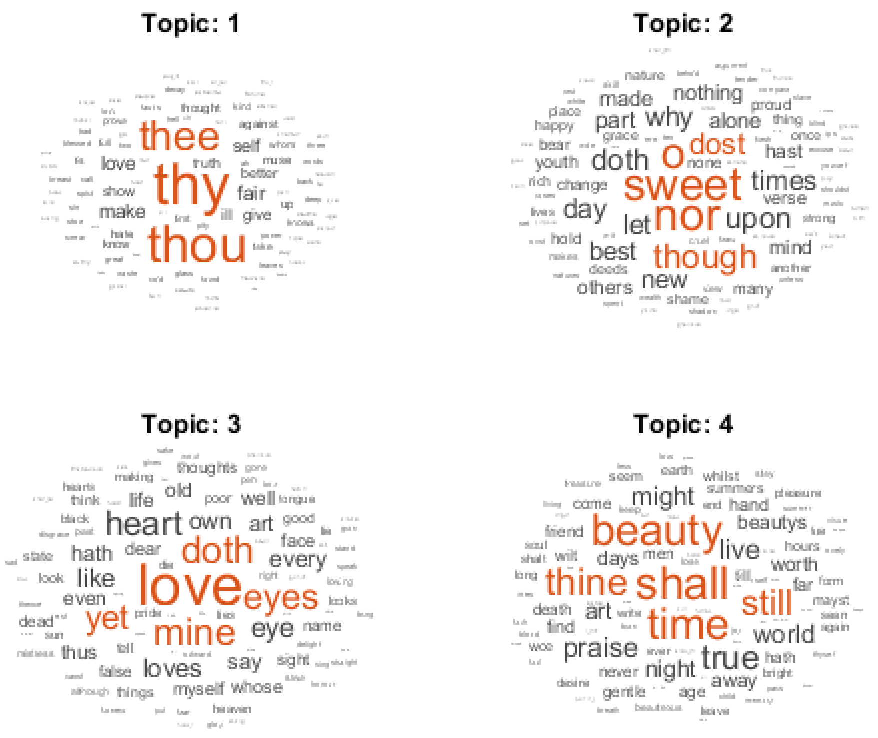
Fit an LDA model with four topics.

```
numTopics = 4;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0887969 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in |  perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.01 |            |  1.215e+03 |         1.000 |             0 |
|          1 |       0.02 | 1.0482e-02 |  1.128e+03 |         1.000 |             0 |
|          2 |       0.02 | 1.7190e-03 |  1.115e+03 |         1.000 |             0 |
|          3 |       0.02 | 4.3796e-04 |  1.118e+03 |         1.000 |             0 |
|          4 |       0.02 | 9.4193e-04 |  1.111e+03 |         1.000 |             0 |
|          5 |       0.02 | 3.7079e-04 |  1.108e+03 |         1.000 |             0 |
|          6 |       0.02 | 9.5777e-05 |  1.107e+03 |         1.000 |             0 |
=====================================================================================

mdl = 
  ldaModel with properties:

                     NumTopics: 4
             WordConcentration: 1
            TopicConcentration: 1
      CorpusTopicProbabilities: [0.2500 0.2500 0.2500 0.2500]
    DocumentTopicProbabilities: [154x4 double]
        TopicWordProbabilities: [3092x4 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Visualize the topics using word clouds.

```
figure
for topicIdx = 1:4
    subplot(2,2,topicIdx)
    wordcloud(mdl,topicIdx);
    title("Topic: " + topicIdx)
end
```



Fit LDA Model to Word Count Matrix

Fit an LDA model to a collection of documents represented by a word count matrix.

To reproduce the results of this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. `sonnetsCounts.mat` contains a matrix of word counts and a corresponding vocabulary of preprocessed versions of Shakespeare's sonnets. The value `counts(i,j)` corresponds to the number of times the `j`th word of the vocabulary appears in the `i`th document.

```
load sonnetsCounts.mat
size(counts)
```

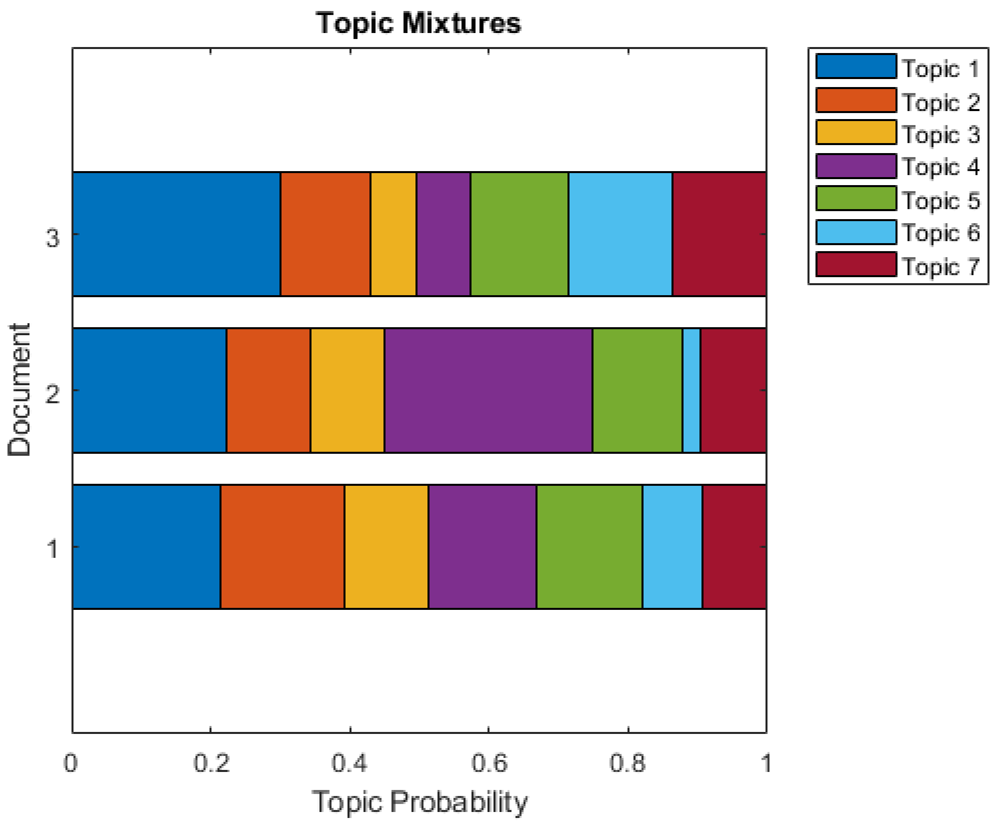
```
ans = 1×2

         154        3092
```

Fit an LDA model with 7 topics. To suppress the verbose output, set `'Verbose'` to 0.

```
numTopics = 7;
mdl = fitlda(counts,numTopics,'Verbose',0);
```

Visualize multiple topic mixtures using stacked bar charts. Visualize the topic mixtures of the first three input documents.

```
topicMixtures = transform(mdl,counts(1:3,:));
figure
barh(topicMixtures,'stacked')
xlim([0 1])
title("Topic Mixtures")
xlabel("Topic Probability")
ylabel("Document")
legend("Topic "+ string(1:numTopics),'Location','northeastoutside')
```



Predict Top LDA Topics of Documents

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LDA model with 20 topics.

```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0612641 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.26 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.06 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.06 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.06 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.06 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.06 | 6.4180e-05 |  7.291e+02 |         5.000 |             0 |
=====================================================================================
```

```
mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

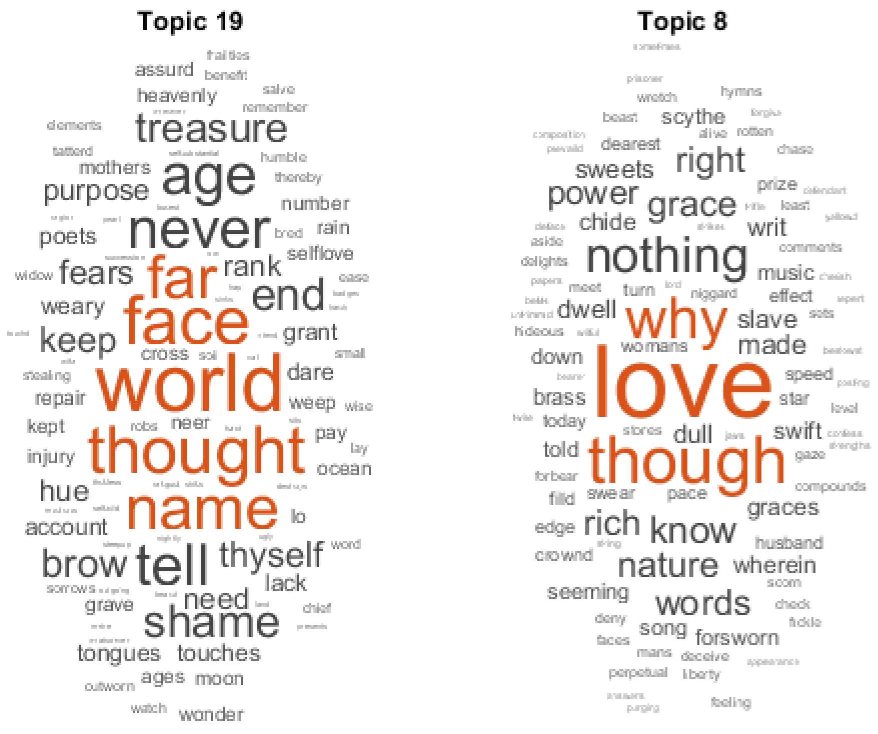
Predict the top topics for an array of new documents.

```
newDocuments = tokenizedDocument([
    "what's in a name? a rose by any other name would smell as sweet."
    "if music be the food of love, play on."]);
topicIdx = predict(mdl,newDocuments)
```

```
topicIdx = 2×1

    19
     8
```

Visualize the predicted topics using word clouds.

```
figure
subplot(1,2,1)
wordcloud(mdl,topicIdx(1));
title("Topic " + topicIdx(1))
subplot(1,2,2)
wordcloud(mdl,topicIdx(2));
title("Topic " + topicIdx(2))
```



Input Arguments

bag — Input model
bagOfWords object | bagOfNgrams object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`numTopics` — Number of topics
positive integer

Number of topics, specified as a positive integer. For an example showing how to choose the number of topics, see “Choose Number of Topics for LDA Model”.

Example: 200

`counts` — Frequency counts of words
matrix of nonnegative integers

Frequency counts of words, specified as a matrix of nonnegative integers. If you specify `'DocumentsIn'` to be `'rows'`, then the value `counts(i,j)` corresponds to the number of times the *j*th word of the vocabulary appears in the *i*th document. Otherwise, the value `counts(i,j)` corresponds to the number of times the *i*th word of the vocabulary appears in the *j*th document.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Solver','avb'` specifies to use approximate variational Bayes as the solver.

Solver Options

`Solver` — Solver for optimization
`'cgs'` (default) | `'savb'` | `'avb'` | `'cvb0'`

Solver for optimization, specified as the comma-separated pair consisting of `'Solver'` and one of the following:

Stochastic Solver

- `'savb'` – Use stochastic approximate variational Bayes [1] [2]. This solver is best suited for large datasets and can fit a good model in fewer passes of the data.

Batch Solvers

- `'cgs'` – Use collapsed Gibbs sampling [3]. This solver can be more accurate at the cost of taking longer to run. The `resume` function does not support models fitted with CGS.
- `'avb'` – Use approximate variational Bayes [4]. This solver typically runs more quickly than collapsed Gibbs sampling and collapsed variational Bayes, but can be less accurate.
- `'cvb0'` – Use collapsed variational Bayes, zeroth order [4] [5]. This solver can be more accurate than approximate variational Bayes at the cost of taking longer to run.

For an example showing how to compare solvers, see “Compare LDA Solvers”.

Example: `'Solver','savb'`

`LogLikelihoodTolerance` — Relative tolerance on log-likelihood
`0.0001` (default) | positive scalar

Relative tolerance on log-likelihood, specified as the comma-separated pair consisting of `'LogLikelihoodTolerance'` and a positive scalar. The optimization terminates when this tolerance is reached.

Example: `'LogLikelihoodTolerance',0.001`

`FitTopicProbabilities` — Option for fitting corpus topic probabilities
`true` (default) | `false`

Option for fitting topic concentration, specified as the comma-separated pair consisting of `'FitTopicConcentration'` and either `true` or `false`.

The function fits the Dirichlet prior $\alpha = \alpha_0(p_1 \; p_2 \; \cdots \; p_K)$ on the topic mixtures, where α_0 is the topic concentration and $p_1, \ldots, p_K$ are the corpus topic probabilities which sum to 1.

Example: `'FitTopicProbabilities',false`

Data Types: `logical`

`FitTopicConcentration` — Option for fitting topic concentration
`true` | `false`

Option for fitting topic concentration, specified as the comma-separated pair consisting of `'FitTopicConcentration'` and either `true` or `false`.

For batch the solvers `'cgs'`, `'avb'`, and `'cvb0'`, the default for `FitTopicConcentration` is `true`. For the stochastic solver `'savb'`, the default is `false`.

The function fits the Dirichlet prior $\alpha = \alpha_0(p_1 \; p_2 \; \cdots \; p_K)$ on the topic mixtures, where α_0 is the topic concentration and $p_1, \ldots, p_K$ are the corpus topic probabilities which sum to 1.

Example: `'FitTopicConcentration',false`

Data Types: `logical`

`InitialTopicConcentration` — Initial estimate of the topic concentration
`numTopics/4` (default) | nonnegative scalar

Initial estimate of the topic concentration, specified as the comma-separated pair consisting of `'InitialTopicConcentration'` and a nonnegative scalar. The function sets the concentration per topic to `TopicConcentration/NumTopics`. For more information, see “Latent Dirichlet Allocation” on page 1-166.

Example: `'InitialTopicConcentration',25`

`TopicOrder` — Topic Order
`'initial-fit-probability'` (default) | `'unordered'`

Topic order, specified as one of the following:

- `'initial-fit-probability'` – Sort the topics by the corpus topic probabilities of input document set (the `CorpusTopicProbabilities` property).
- `'unordered'` – Do not sort the topics.

`WordConcentration` — Word concentration
`1` (default) | nonnegative scalar

Word concentration, specified as the comma-separated pair consisting of `'WordConcentration'` and a nonnegative scalar. The software sets the Dirichlet prior on the topics (the word probabilities per topic) to be the symmetric Dirichlet distribution parameter with the value `WordConcentration/numWords`, where `numWords` is the vocabulary size of the input documents. For more information, see “Latent Dirichlet Allocation” on page 1-166.

`DocumentsIn` — Orientation of documents
`'rows'` (default) | `'columns'`

Orientation of documents in the word count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Input is a matrix of word counts with rows corresponding to documents.
- `'columns'` – Input is a transposed matrix of word counts with columns corresponding to documents.

This option only applies if you specify the input documents as a matrix of word counts.

Note If you orient your word count matrix so that documents correspond to columns and specify `'DocumentsIn','columns'`, then you might experience a significant reduction in optimization-execution time.

Batch Solver Options

`IterationLimit` — Maximum number of iterations
`100` (default) | positive integer

Maximum number of iterations, specified as the comma-separated pair consisting of `'IterationLimit'` and a positive integer.

This option supports batch solvers only (`'cgs'`, `'avb'`, or `'cvb0'`).

Example: `'IterationLimit',200`

Stochastic Solver Options

`DataPassLimit` — Maximum number of passes through data
1 (default) | positive integer

Maximum number of passes through the data, specified as the comma-separated pair consisting of `'DataPassLimit'` and a positive integer.

If you specify `'DataPassLimit'` but not `'MiniBatchLimit'`, then the default value of `'MiniBatchLimit'` is ignored. If you specify both `'DataPassLimit'` and `'MiniBatchLimit'`, then `fitlda` uses the argument that results in processing the fewest observations.

This option supports only the stochastic (`'savb'`) solver.

Example: `'DataPassLimit',2`

`MiniBatchLimit` — Maximum number of mini-batch passes
positive integer

Maximum number of mini-batch passes, specified as the comma-separated pair consisting of `'MiniBatchLimit'` and a positive integer.

If you specify `'MiniBatchLimit'` but not `'DataPassLimit'`, then `fitlda` ignores the default value of `'DataPassLimit'`. If you specify both `'MiniBatchLimit'` and `'DataPassLimit'`, then `fitlda` uses the argument that results in processing the fewest observations. The default value is `ceil(numDocuments/MiniBatchSize)`, where `numDocuments` is the number of input documents.

This option supports only the stochastic (`'savb'`) solver.

Example: `'MiniBatchLimit',200`

`MiniBatchSize` — Mini-batch size
1000 (default) | positive integer

Mini-batch size, specified as the comma-separated pair consisting of `'MiniBatchLimit'` and a positive integer. The function processes `MiniBatchSize` documents in each iteration.

This option supports only the stochastic (`'savb'`) solver.

Example: `'MiniBatchSize',512`

`LearnRateDecay` — Learning rate decay
0.5 (default) | positive scalar less than or equal to 1

Learning rate decay, specified as the comma-separated pair `'LearnRateDecay'` and a positive scalar less than or equal to 1.

For mini-batch t, the function sets the learning rate to $\eta(t) = 1/(1 + t)^{\kappa}$, where κ is the learning rate decay.

If `LearnRateDecay` is close to 1, then the learning rate decays faster and the model learns mostly from the earlier mini-batches. If `LearnRateDecay` is close to 0, then the learning rate decays slower and the model continues to learn from more mini-batches. For more information, see “Stochastic Solver” on page 1-167.

This option supports the stochastic solver only (`'savb'`).

Example: `'LearnRateDecay',0.75`

Display Options

`ValidationData` — Validation data
`[]` (default) | `bagOfWords` object | `bagOfNgrams` object | sparse matrix of word counts

Validation data to monitor optimization convergence, specified as the comma-separated pair consisting of `'ValidationData'` and a `bagOfWords` object, a `bagOfNgrams` object, or a sparse matrix of word counts. If the validation data is a matrix, then the data must have the same orientation and the same number of words as the input documents.

`ValidationFrequency` — Frequency of model validation
positive integer

Frequency of model validation in number of iterations, specified as the comma-separated pair consisting of `'ValidationFrequency'` and a positive integer.

The default value depends on the solver used to fit the model. For the stochastic solver, the default value is 10. For the other solvers, the default value is 1.

`Verbose` — Verbosity level
1 (default) | 0

Verbosity level, specified as the comma-separated pair consisting of `'Verbose'` and one of the following:

- 0 – Do not display verbose output.
- 1 – Display progress information.

Example: `'Verbose',0`

Output Arguments

mdl — Output LDA model
ldaModel object

Output LDA model, returned as an `ldaModel` object.

More About

Latent Dirichlet Allocation

A *latent Dirichlet allocation* (LDA) model is a document topic model which discovers underlying topics in a collection of documents and infers word probabilities in topics. LDA models a collection of D documents as topic mixtures $\theta_1, \ldots, \theta_D$, over K topics characterized by vectors of word probabilities $\varphi_1, \ldots, \varphi_K$. The model assumes that the topic mixtures $\theta_1, \ldots, \theta_D$, and the topics $\varphi_1, \ldots, \varphi_K$ follow a Dirichlet distribution with concentration parameters α and β respectively.

The topic mixtures $\theta_1, \ldots, \theta_D$ are probability vectors of length K, where K is the number of topics. The entry θ_{di} is the probability of topic i appearing in the dth document. The topic mixtures correspond to the rows of the `DocumentTopicProbabilities` property of the `ldaModel` object.

The topics $\varphi_1, \ldots, \varphi_K$ are probability vectors of length V, where V is the number of words in the vocabulary. The entry φ_{iv} corresponds to the probability of the vth word of the vocabulary appearing in the ith topic. The topics $\varphi_1, \ldots, \varphi_K$ correspond to the columns of the `TopicWordProbabilities` property of the `ldaModel` object.

Given the topics $\varphi_1, \ldots, \varphi_K$ and Dirichlet prior α on the topic mixtures, LDA assumes the following generative process for a document:

1. Sample a topic mixture $\theta \sim \text{Dirichlet}(\alpha)$. The random variable θ is a probability vector of length K, where K is the number of topics.
2. For each word in the document:
   - a. Sample a topic index $z \sim \text{Categorical}(\theta)$. The random variable z is an integer from 1 through K, where K is the number of topics.
   - b. Sample a word $w \sim \text{Categorical}(\varphi_z)$. The random variable w is an integer from 1 through V, where V is the number of words in the vocabulary, and represents the corresponding word in the vocabulary.

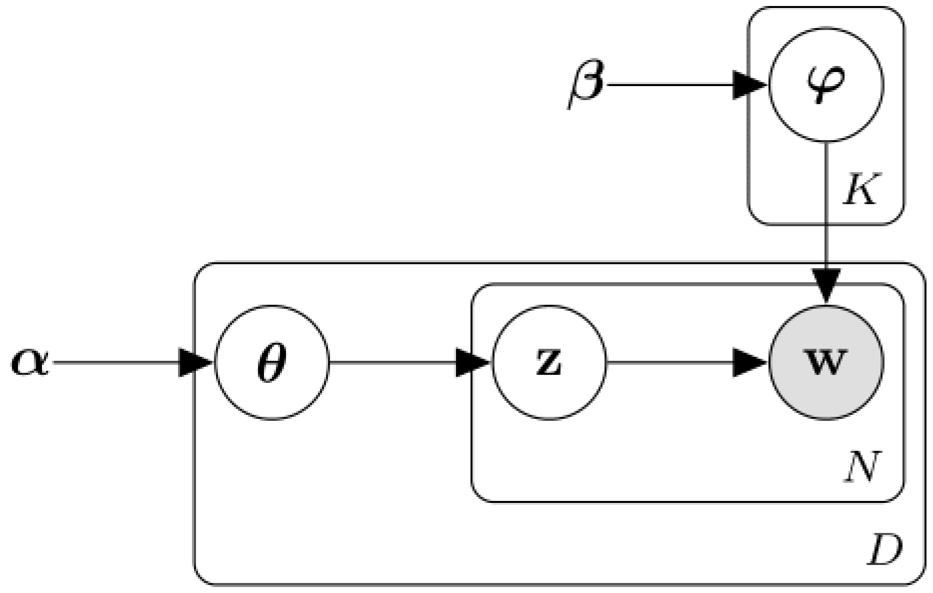
Under this generative process, the joint distribution of a document with words $w_1, \ldots, w_N$, with topic mixture θ, and with topic indices $z_1, \ldots, z_N$ is given by

$$p(\theta, z, w \,\Big|\, \alpha, \varphi) = p(\theta \,\Big|\, \alpha) \prod_{n=1}^{N} p(z_n \,\Big|\, \theta) p(w_n \,\Big|\, z_n, \varphi),$$

where N is the number of words in the document. Summing the joint distribution over z and then integrating over θ yields the marginal distribution of a document w:

$$p(w \,\Big|\, \alpha, \varphi) = \int_{\theta} p(\theta \,\Big|\, \alpha) \prod_{n=1}^{N} \sum_{z_n} p(z_n | \theta) p(w_n | z_n, \varphi) d\theta\,.$$

The following diagram illustrates the LDA model as a probabilistic graphical model. Shaded nodes are observed variables, unshaded nodes are latent variables, nodes without outlines are the model parameters. The arrows highlight dependencies between random variables and the plates indicate repeated nodes.



Dirichlet Distribution

The *Dirichlet distribution* is a continuous generalization of the multinomial distribution. Given the number of categories $K \geq 2$, and concentration parameter α, where α is a vector of positive reals of length K, the probability density function of the Dirichlet distribution is given by

$$p(\theta \mid \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} \theta_i^{\alpha_i - 1},$$

where B denotes the multivariate Beta function given by

$$B(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}.$$

A special case of the Dirichlet distribution is the *symmetric Dirichlet distribution*. The symmetric Dirichlet distribution is characterized by the concentration parameter α, where all the elements of α are the same.

Stochastic Solver

The stochastic solver processes documents in mini-batches. It updates the per-topic word probabilities using a weighted sum of the probabilities calculated from each mini-batch, and the probabilities from all previous mini-batches.

For mini-batch t, the solver sets the learning rate to $\eta(t) = 1/(1 + t)^{\kappa}$, where κ is the learning rate decay.

The function uses the learning rate decay to update Φ, the matrix of word probabilities per topic, by setting

$$\Phi^{(t)} = (1 - \eta(t))\Phi^{(t-1)} + \eta(t)\Phi^{(t*)},$$

where $\Phi^{(t*)}$ is the matrix learned from mini-batch t, and $\Phi^{(t-1)}$ is the matrix learned from mini-batches 1 through t-1.

Before learning begins (when $t = 0$), the function initializes the initial word probabilities per topic $\Phi^{(0)}$ with random values.

Compatibility Considerations

`fitlda` sorts topics

Behavior changed in R2018b

Starting in R2018b, `fitlda`, by default, sorts the topics in descending order of the topic probabilities of the input document set. This behavior makes it easier to find the topics with the highest probabilities.

In previous versions, `fitlda` does not change the topic order. To reproduce the behavior, set the `'TopicOrder'` option to `'unordered'`.

References

[1] Foulds, James, Levi Boyles, Christopher DuBois, Padhraic Smyth, and Max Welling. "Stochastic collapsed variational Bayesian inference for latent Dirichlet allocation." In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 446–454. ACM, 2013.

[2] Hoffman, Matthew D., David M. Blei, Chong Wang, and John Paisley. "Stochastic variational inference." *The Journal of Machine Learning Research* 14, no. 1 (2013): 1303–1347.

[3] Griffiths, Thomas L., and Mark Steyvers. "Finding scientific topics." *Proceedings of the National academy of Sciences* 101, no. suppl 1 (2004): 5228–5235.

[4] Asuncion, Arthur, Max Welling, Padhraic Smyth, and Yee Whye Teh. "On smoothing and inference for topic models." In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 27–34. AUAI Press, 2009.

[5] Teh, Yee W., David Newman, and Max Welling. "A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation." In *Advances in neural information processing systems*, pp. 1353–1360. 2007.

See Also

`bagOfNgrams` | `bagOfWords` | `fitlsa` | `ldaModel` | `logp` | `lsaModel` | `predict` | `resume` | `topkwords` | `transform` | `wordcloud`

Topics
“Analyze Text Data Using Topic Models”
“Choose Number of Topics for LDA Model”
“Compare LDA Solvers”
“Analyze Text Data Using Multiword Phrases”
“Classify Text Data Using Deep Learning”

Introduced in R2017b

fitlsa

Fit LSA model

Syntax

```
mdl = fitlsa(bag,numComponents)
mdl = fitlsa(counts,numComponents)
mdl = fitlsa( ___ ,Name,Value)
```

Description

A latent semantic analysis (LSA) model discovers relationships between documents and the words that they contain. An LSA model is a dimensionality reduction tool useful for running low-dimensional statistical models on high-dimensional word counts. If the model was fit using a bag-of-n-grams model, then the software treats the n-grams as individual words.

`mdl = fitlsa(bag,numComponents)` fits an LSA model with `numComponents` components to the bag-of-words or bag-of-n-grams model `bag`.

`mdl = fitlsa(counts,numComponents)` fits an LSA model to the documents represented by the matrix of word counts `counts`.

`mdl = fitlsa( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Fit LSA Model

Fit a Latent Semantic Analysis model to a collection of documents.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
```

```
        NumWords: 3092
    NumDocuments: 154
```

Fit an LSA model with 20 components.

```
numComponents = 20;
mdl = fitlsa(bag,numComponents)
```

```
mdl =
  lsaModel with properties:

               NumComponents: 20
            ComponentWeights: [1x20 double]
              DocumentScores: [154x20 double]
                  WordScores: [3092x20 double]
                  Vocabulary: [1x3092 string]
    FeatureStrengthExponent: 2
```

Transform new documents into lower dimensional space using the LSA model.

```
newDocuments = tokenizedDocument([
    "what's in a name? a rose by any other name would smell as sweet."
    "if music be the food of love, play on."]);
dscores = transform(mdl,newDocuments)
```

```
dscores = 2×20

    0.1338    0.1623    0.1680   -0.0541   -0.2464   -0.0134    0.2604   -0.0205   -0.1127    0.0
    0.2547    0.5576   -0.0095    0.5660   -0.0643   -0.1236   -0.0082    0.0522    0.0690   -0.0
```

Fit LSA Model to Word Count Matrix

Load the example data. `sonnetsCounts.mat` contains a matrix of word counts corresponding to preprocessed versions of Shakespeare's sonnets.

```
load sonnetsCounts.mat
size(counts)
```

```
ans = 1×2

         154        3092
```

Fit LSA model with 20 components. Set the feature strength exponent to 4.

```
numComponents = 20;
exponent = 4;
mdl = fitlsa(counts,numComponents, ...
    'FeatureStrengthExponent',exponent)
```

```
mdl =
  lsaModel with properties:
```

```
                NumComponents: 20
             ComponentWeights: [1x20 double]
               DocumentScores: [154x20 double]
                   WordScores: [3092x20 double]
                   Vocabulary: [1x3092 string]
    FeatureStrengthExponent: 4
```

Input Arguments

bag — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`numComponents` — Number of components
positive integer

Number of components, specified as a positive integer. This value must be less than the number of the input documents, and the vocabulary size of the input documents.

Example: 200

`counts` — Frequency counts of words
matrix of nonnegative integers

Frequency counts of words, specified as a matrix of nonnegative integers. If you specify `'DocumentsIn'` to be `'rows'`, then the value `counts(i,j)` corresponds to the number of times the *j*th word of the vocabulary appears in the *i*th document. Otherwise, the value `counts(i,j)` corresponds to the number of times the *i*th word of the vocabulary appears in the *j*th document.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'FeatureStrengthExponent',4` sets the feature strength exponent to 4.

`DocumentsIn` — Orientation of documents
`'rows'` (default) | `'columns'`

Orientation of documents in the word count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Input is a matrix of word counts with rows corresponding to documents.
- `'columns'` – Input is a transposed matrix of word counts with columns corresponding to documents.

This option only applies if you specify the input documents as a matrix of word counts.

Note If you orient your word count matrix so that documents correspond to columns and specify `'DocumentsIn','columns'`, then you might experience a significant reduction in optimization-execution time.

`FeatureStrengthExponent` — Initial feature strength exponent
2 (default) | nonnegative scalar

Initial feature strength exponent, specified as a nonnegative scalar. This value scales the feature component strengths for the `documentScores`, `wordScores`, and `transform` functions.

Example: `'FeatureStrengthExponent',4`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`mdl` — Output LSA model
`lsaModel` object

Output LSA model, returned as an `lsaModel` object.

See Also
`bagOfNgrams` | `bagOfWords` | `fitlda` | `ldaModel` | `lsaModel` | `transform`

Topics
"Analyze Text Data Using Topic Models"
"Choose Number of Topics for LDA Model"
"Compare LDA Solvers"
"Analyze Text Data Using Multiword Phrases"
"Classify Text Data Using Deep Learning"

Introduced in R2017b

getAttribute

Read HTML attribute of root node of HTML tree

Syntax

```
str = getAttribute(tree,attr)
```

Description

`str = getAttribute(tree,attr)` returns the attribute `attr` of the root node of `tree`. If that attribute is not set, then the function returns a `missing` value.

Examples

Get Attribute of HTML Tag

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using `webread`.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

Find all the hyperlinks in the HTML tree using `findElement`. The hyperlinks are the nodes with element name `"A"`.

```
selector = "A";
subtrees = findElement(tree,selector);
subtrees(1:10)
```

```
ans = 
  10×1 htmlTree:

    <A class="svg_link navbar-brand" href="https://www.mathworks.com?s_tid=gn_logo"><IMG alt="Ma
    <A class="mwa-nav_login" href="https://www.mathworks.com/login?uri=http://www.mathworks.com/
    <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Products</A>
    <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solutions</A>
    <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Academia</A>
    <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Support</A>
    <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Community</A>
    <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">Events</A>
    <A href="https://www.mathworks.com/company/aboutus/contact_us.html?s_tid=gn_cntus">Contact U
    <A href="https://www.mathworks.com/store?s_cid=store_top_nav&amp;s_tid=gn_store">How to Buy<
```

Get the hyperlink references using `getAttribute`. Specify the attribute name `"href"`.

```
attr = "href";
str = getAttribute(subtrees,attr);
str(1:10)
```

```
ans = 10×1 string array
    "https://www.mathworks.com?s_tid=gn_logo"
    "https://www.mathworks.com/login?uri=http://www.mathworks.com/help/textanalytics/index.html"
    "https://www.mathworks.com/products.html?s_tid=gn_ps"
    "https://www.mathworks.com/solutions.html?s_tid=gn_sol"
    "https://www.mathworks.com/academia.html?s_tid=gn_acad"
    "https://www.mathworks.com/support.html?s_tid=gn_supp"
    "https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc"
    "https://www.mathworks.com/company/events.html?s_tid=gn_ev"
    "https://www.mathworks.com/company/aboutus/contact_us.html?s_tid=gn_cntus"
    "https://www.mathworks.com/store?s_cid=store_top_nav&s_tid=gn_store"
```

Input Arguments

`tree` — HTML tree
`htmlTree` array

HTML tree, specified as an `htmlTree` array.

`attr` — Attribute name
string scalar | character vector | scalar cell array containing a character vector

Attribute name, specified as a string scalar, character vector, or a scalar cell array containing a character vector.

Output Arguments

`str` — HTML attribute
string array

HTML attribute, returned as a string array

More About

HTML Elements

A typical HTML element contains the following components:

- Element name – Name of the HTML tag. The element name corresponds to the `Name` property of the HTML tree.
- Attributes – Additional information about the tag. HTML attributes have the form `*name*="*value*"`, where *`name`* and *`value`* denote the attribute name and value respectively. The attributes appear inside the opening HTML tag. To get the attribute values from an HTML tree, use `getAttribute`.
- Content – Element content. The content appears between opening and closing HTML tags. The content can be text data or nested HTML elements. To extract the text from an `htmlTree` object, use `extractHTMLText`. To get the nested HTML elements of an `htmlTree` object, use the `Children` property.

For example, the HTML element `<a href="https://www.mathworks.com">Home</a>` comprises the following components:

Component		Value	Description
Element name		`a`	Element is a hyperlink
Attribute	Attribute name	`href`	Hyperlink reference
	Attribute value	`"https://www.mathworks.com"`	Hyperlink reference value
Content		`Home`	Text to display

See Also

`extractFileText` | `extractHTMLText` | `findElement` | `htmlTree` | `ismissing` | `readPDFFormData` | `tokenizedDocument`

Topics

"Parse HTML and Extract Text Content"
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018b

hex

Package: `textanalytics.unicode`

Convert UTF-32 representation to hexadecimal values

Syntax

```
hexStr = hex(str32)
```

Description

`hexStr = hex(str32)` converts the UTF-32 representation `str32` to hexadecimal values.

Examples

Convert UTF-32 String Representation to Hexadecimal Values

Convert the string `"Hello! 😀"` to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! 😀";
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Convert `str32` to hexadecimal values using the `hex` function.

```
hexStr = hex(str32)
```

```
hexStr = 
" 0048  0065  006C  006C  006F  0021  0020 1F600"
```

Input Arguments

`str32` — UTF-32 string representation
UTF32 array

UTF-32 string representation, specified as a `UTF32` array.

Output Arguments

`hexStr` — Hexadecimal values
string array

Hexadecimal values, returned as a string array.

The output `hexStr` is a string array with the same size as `str32`, where `hexStr(i)` is a string containing the hexadecimal values corresponding to the `str32(i)` separated by whitespace characters.

See Also

`characterCategories` | `textanalytics.unicode.nfd` | `tokenizedDocument`

Topics

"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Language Considerations"

Introduced in R2021a

htmlTree

Parsed HTML tree

Description

An `htmlTree` object represents a parsed HTML element or node. Extract parts of interest using the `findElement` function or the `Children` property, and extract text using the `extractHTMLText` function.

Creation

Syntax

`tree = htmlTree(code)`

Description

`tree = htmlTree(code)` parses the HTML code in the string `code` and returns the resulting tree structure.

Input Arguments

code — HTML code
string array | character vector | cell array of character vectors

HTML code, specified as a string array, a character vector, or a cell array of character vectors.

Tip

- To read HTML code from a web page, use `webread`.
- To extract text from an HTML file, use `extractFileText`.

Example: `"<a href='https://www.mathworks.com'>MathWorks</a>"`

Data Types: `char` | `string` | `cell`

Properties

`Children` — Direct descendants of element
`htmlTree` array

Direct descendants of the element, specified as an `htmlTree` array.

`Parent` — Parent node
`htmlTree` object

Parent node in the tree, specified as an `htmlTree` object.

If the HTML tree is a root node, then the value of `Parent` is `missing`.

Name — HTML element name
string scalar

HTML element name, specified as a string scalar.

For more information, see "HTML Elements" on page 1-184.

Object Functions

findElement	Find elements in HTML tree
getAttribute	Read HTML attribute of root node of HTML tree
extractHTMLText	Extract text from HTML
ismissing	Find HTML trees without values

Examples

Parse HTML Code

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using `webread`.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

View the element name of the root node of the tree.

```
tree.Name
```

```
ans = 
"HTML"
```

View the children of the root node.

```
tree.Children
```

```
ans = 
  4×1 htmlTree:

    " "
    <HEAD><TITLE>Text Analytics Toolbox Documentation</TITLE><META charset="utf-8"/><META conten
    " "
    <BODY id="responsive_offcanvas"><!-- Mobile TopNav: Start --><DIV class="header visible-xs vi
```

Extract the text from the HTML tree using `extractHTMLText`.

```
str = extractHTMLText(tree)
str = 
    "Text Analytics Toolbox™ provides algorithms and visualizations for preprocessing, analyzing

     Text Analytics Toolbox includes tools for processing raw text from sources such as equipment

     Using machine learning techniques such as LSA, LDA, and word embeddings, you can find cluste
```

Find Elements in HTML Tree

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using the `webread` function.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

Find all the hyperlinks in the HTML tree using `findElement`. The hyperlinks are nodes with element name `"A"`.

```
selector = "A";
subtrees = findElement(tree,selector);
```

View the first few subtrees.

```
subtrees(1:10)
```

```
ans = 
  10×1 htmlTree:

    <A class="skip_link sr-only" href="#content_container">Skip to content</A>
    <A href="https://www.mathworks.com?s_tid=gn_logo" class="svg_link navbar-brand"><IMG src="/im
    <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Products</A>
    <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solutions</A>
    <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Academia</A>
    <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Support</A>
    <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Community</A>
    <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">Events</A>
    <A href="https://www.mathworks.com/products/get-matlab.html?s_tid=gn_getml">Get MATLAB</A>
    <A href="https://www.mathworks.com?s_tid=gn_logo" class="svg_link pull-left"><IMG src="/image
```

Extract the text from the subtrees using `extractHTMLText`. The result contains the link text from each link on the page.

```
str = extractHTMLText(subtrees);
str(1:10)
```

```
ans = 10×1 string
    "Skip to content"
    ""
    "Products"
    "Solutions"
    "Academia"
    "Support"
    "Community"
    "Events"
    "Get MATLAB"
    ""
```

Get Attribute of HTML Tag

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using `webread`.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

Find all the hyperlinks in the HTML tree using `findElement`. The hyperlinks are the nodes with element name `"A"`.

```
selector = "A";
subtrees = findElement(tree,selector);
subtrees(1:10)
```

```
ans = 
  10×1 htmlTree:

    <A class="svg_link navbar-brand" href="https://www.mathworks.com?s_tid=gn_logo"><IMG alt="Ma
    <A class="mwa-nav_login" href="https://www.mathworks.com/login?uri=http://www.mathworks.com/
    <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Products</A>
    <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solutions</A>
    <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Academia</A>
    <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Support</A>
    <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Community</A>
    <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">Events</A>
    <A href="https://www.mathworks.com/company/aboutus/contact_us.html?s_tid=gn_cntus">Contact U
    <A href="https://www.mathworks.com/store?s_cid=store_top_nav&amp;s_tid=gn_store">How to Buy<
```

Get the hyperlink references using `getAttribute`. Specify the attribute name `"href"`.

```
attr = "href";
str = getAttribute(subtrees,attr);
str(1:10)
```

```
ans = 10×1 string array
    "https://www.mathworks.com?s_tid=gn_logo"
    "https://www.mathworks.com/login?uri=http://www.mathworks.com/help/textanalytics/index.html"
    "https://www.mathworks.com/products.html?s_tid=gn_ps"
    "https://www.mathworks.com/solutions.html?s_tid=gn_sol"
    "https://www.mathworks.com/academia.html?s_tid=gn_acad"
    "https://www.mathworks.com/support.html?s_tid=gn_supp"
    "https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc"
    "https://www.mathworks.com/company/events.html?s_tid=gn_ev"
    "https://www.mathworks.com/company/aboutus/contact_us.html?s_tid=gn_cntus"
    "https://www.mathworks.com/store?s_cid=store_top_nav&s_tid=gn_store"
```

Convert Parsed HTML Code to String

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using the `webread` function.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using the `htmlTree` function.

```
tree = htmlTree(code);
```

Find all the paragraphs in the HTML tree using the `findElement` function. The paragraphs are the nodes with element name "P".

```
subtrees = findElement(tree,"P");
```

Convert the subtrees to string using the `string` function.

```
str = string(subtrees)
```

```
str = 26×1 string
    "<P class="h1">↵  <A href="../index.html" class="coming_from_product">Documentation</A>↵  <A
    "<P>Text Analytics Toolbox™ provides algorithms and visualizations for preprocessing, analyz
    "<P>Text Analytics Toolbox includes tools for processing raw text from sources such as equipm
    "<P>Using machine learning techniques such as LSA, LDA, and word embeddings, you can find cl
    "<P class="category_desc">Learn the basics of Text Analytics Toolbox</P>"
    "<P class="category_desc">Import text data into MATLAB<SUP>®</SUP> and preprocess it for ana
    "<P class="category_desc">Develop predictive models using topic models and word embeddings</
    "<P class="category_desc">Visualize text data and models using word clouds and text scatter 
    "<P class="category_desc">Information on language support in Text Analytics Toolbox</P>"
    "<P>You clicked a link that corresponds to this MATLAB command:</P>"
    "<P>Run the command by entering it in the MATLAB Command Window. Web browsers do not support
    "<P>Choose a web site to get translated content where available and see local events and off
    "<P>You can also select a web site from the following list:</P>"
    "<P>Select the China site (in Chinese or English) for best site performance. Other MathWorks
    "<P class="text-center">↵  <A href="#" class="worldwide_link">Contact your local office</A>↵
    "<P class="ff_section_title">Explore Products</P>"
    "<P class="ff_section_title">Try or Buy</P>"
    "<P class="ff_section_title">Learn to Use</P>"
    "<P class="ff_section_title">Get Support</P>"
    "<P class="ff_section_title">About <SPAN translate="no">MathWorks</SPAN></P>"
    "<P class="h4 add_font_futura_medium add_margin_0">↵  <SPAN translate="no">MathWorks</SPAN>↵
    "<P>↵  <EM>Accelerating the pace of engineering and science</EM>↵</P>"
    "<P><SPAN translate="no">MathWorks</SPAN> is the leading developer of mathematical computing 
    "<P>↵  <A href="/discovery.html?s_tid=all_disc_mw_ff">Discover...</A>↵</P>"
    "<P class="copyright" translate="no">© 1994-2021 The MathWorks, Inc.</P>"
    "<P>↵  <EM>Join the conversation</EM>↵</P>"
```

More About

HTML Elements

A typical HTML element contains the following components:

- Element name – Name of the HTML tag. The element name corresponds to the `Name` property of the HTML tree.
- Attributes – Additional information about the tag. HTML attributes have the form *`name`*`="`*`value`*`"`, where *`name`* and *`value`* denote the attribute name and value respectively. The attributes appear inside the opening HTML tag. To get the attribute values from an HTML tree, use `getAttribute`.
- Content – Element content. The content appears between opening and closing HTML tags. The content can be text data or nested HTML elements. To extract the text from an `htmlTree` object, use `extractHTMLText`. To get the nested HTML elements of an `htmlTree` object, use the `Children` property.

For example, the HTML element `<a href="https://www.mathworks.com">Home</a>` comprises the following components:

Component		Value	Description
Element name		`a`	Element is a hyperlink
Attribute	Attribute name	`href`	Hyperlink reference
	Attribute value	`"https://www.mathworks.com"`	Hyperlink reference value
Content		`Home`	Text to display

Compatibility Considerations

`htmlTree` uses different algorithm for restructuring malformed HTML
Behavior changed in R2021a

When creating an `htmlTree` object, the software automatically restructures malformed input HTML code to have valid structure. This restructuring process includes adding, removing, and editing elements as well as rearranging the tree structure. Starting in R2021a, the software uses an updated algorithm to restructure malformed HTML. This change can result in `htmlTree` objects created in R2021a or later having different size, structure, and content when compared to previous releases.

Starting in R2021a, when loading `htmlTree` objects from MAT files created in an R2020b or before, the software automatically restructures the `htmlTree` object using the same algorithm used for creating `htmlTree` objects. When loading `htmlTree` objects from MAT files created in R2021a or later, the software does not restructure the `htmlTree` object.

This table highlights some notable steps of the restructuring process:

Step	Change in Behavior
Automatically add head and title elements.	Starting in R2021a, when creating an `htmlTree` object from HTML code, the software automatically inserts missing `<HEAD>`, `<TITLE>`, and other elements. In previous versions, the `htmlTree` object only included these elements when they are present in the input code. When loading `htmlTree` objects from MAT files created in an earlier release, the software automatically inserts `<HEAD>` and `<TITLE>` elements. When loading `htmlTree` objects from MAT files created in R2021a or later, the software does not automatically insert these elements.
Automatically add missing elements.	Starting in R2021a, when creating an `htmlTree` object from HTML code, the software automatically inserts missing elements when parent and child elements are inconsistent. For example, when a `<li>` (list item) element does not have a parent `<ul>` (unordered list) or `<ol>` (unordered list) element, the software automatically adds a `<ul>` element to make the HTML valid. This can result in different outputs when compared with earlier releases. When loading `htmlTree` objects from MAT files created in an earlier release, the software automatically inserts missing elements. When loading `htmlTree` objects from MAT files created in R2021a or later, the software does not automatically insert missing elements.
Discard parts of malformed code.	When creating an `htmlTree` object with malformed HTML code, the software may discard parts of the text. For example, if the input code is the string `"<div>a</"`, then the software discards the text `"a</"`.

See Also

`extractHTMLText` | `findElement` | `getAttribute` | `ismissing` | `readPDFFormData` | `tokenizedDocument`

Topics

"Parse HTML and Extract Text Content"
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018b

ind2word

Map encoding index to word

Syntax

```
words = ind2word(enc,M)
```

Description

`words = ind2word(enc,M)` returns the words corresponding to the encoding indices in `M` according to the word encoding `enc`.

Examples

Map Encoding Indices to Words

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
documents(1:10)
```

```
ans = 
  10x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youtl
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair tl
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial t
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lovs
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt haj
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art belo
```

Create a word encoding.

```
enc = wordEncoding(documents)
```

```
enc = 
  wordEncoding with properties:

      NumWords: 3092
    Vocabulary: [1x3092 string]
```

View the words corresponding to indices 1, 3, and 5 using the `ind2word` function.

```
idx = [1 3 5];
words = ind2word(enc,idx)
```

```
words = 1x3 string
    "fairest"    "desire"    "thereby"
```

Input Arguments

`enc` — Input word encoding
`wordEncoding` object

Input word encoding, specified as a `wordEncoding` object.

`M` — Word encoding indices
vector of positive integers

Word encoding indices, specified as a vector of positive integers.

Output Arguments

`words` — Output words
string vector

Output words, returned as a string vector.

See Also

`doc2sequence` | `fastTextWordEmbedding` | `isVocabularyWord` | `tokenizedDocument` | `vec2word` | `word2ind` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding`

Topics
"Train a Sentiment Classifier"
"Classify Text Data Using Deep Learning"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2018b

ismember

(To be removed) Test if word is member of word embedding

Note `ismember` will be removed in a future release. Use `isVocabularyWord` instead. For more information, see "Compatibility Considerations".

Syntax

```
tf = ismember(emb,words)
```

Description

`tf = ismember(emb,words)` returns an array containing logical `1` (`true`) where the word in `words` is a member of the word embedding `emb`. Elsewhere, the array contains logical `0` (`false`).

Examples

Test If Word Is Member of Embedding

Test to determine if words are members of a word embedding.

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb = 

  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

Test if the words `"I"`, `"love"`, and `"fastTextWordEmbedding"` are in the word embedding.

```
words = ["I" "love" "fastTextWordEmbedding"];
tf = ismember(emb,words)
```

```
tf = 

  1×3 logical array

   1   1   0
```

Input Arguments

emb — Input word embedding
`wordEmbedding` object

Input word embedding, specified as a `wordEmbedding` object.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

Compatibility Considerations

`ismember` will be removed
Warns starting in R2018b

To update your code, for `wordEmbedding` object input, change the function name from `ismember` to `isVocabularyWord`. You do not need to change the arguments. The syntaxes are equivalent.

See Also

`fastTextWordEmbedding` | `isVocabularyWord` | `tokenizedDocument` | `vec2word` | `word2vec` | `wordEmbedding`

Topics
“Train a Sentiment Classifier”
“Classify Text Data Using Deep Learning”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2017b

ismissing

Find HTML trees without values

Syntax

```
tf = ismissing(tree)
```

Description

`tf = ismissing(tree)` returns a logical array that indicates which elements of `tree` do not reference HTML trees. For example, if `tree` is given by the `Parent` property of a root node, then the function returns 1 (`true`).

Examples

Test If HTML Tree Is Root Node

To test if an HTML tree object represents a root node, test that the `Parent` property is missing.

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using `webread`.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

Test if the parent of `tree` references an HTML tree.

```
tf = ismissing(tree.Parent)
```

```
tf = logical
   1
```

Since `tree` represents the root node of the HTML tree, the value of `tree.Parent` is `missing` and the `ismissing` function returns 1 (`true`).

Input Arguments

`tree` — HTML tree
`htmlTree` array

HTML tree, specified as an `htmlTree` array.

See Also

`extractFileText` | `extractHTMLText` | `findElement` | `getAttribute` | `htmlTree` | `readPDFFormData` | `tokenizedDocument`

Topics

"Parse HTML and Extract Text Content"
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018b

isVocabularyWord

Test if word is member of word embedding or encoding

Syntax

```
tf = isVocabularyWord(emb,words)
tf = isVocabularyWord(enc,words)
tf = isVocabularyWord( ___ ,'IgnoreCase',true)
```

Description

`tf = isVocabularyWord(emb,words)` tests if the elements of `words` are members of the word embedding `emb`. The function returns a logical array containing `1` (`true`) where the words are members of the word embedding. Elsewhere, the array contains `0` (`false`). The function, by default, is case sensitive.

`tf = isVocabularyWord(enc,words)` tests if the elements of `words` are members of the word encoding `enc`. The function, by default, is case sensitive.

`tf = isVocabularyWord( ___ ,'IgnoreCase',true)` tests if the specified words are in the vocabulary ignoring case using any of the previous syntaxes.

Examples

Test If Word Is Member of Embedding

Test to determine if words are members of a word embedding.

Load a pretrained word embedding using the `fastTextWordEmbedding` function. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×999994 string]
```

Test if the words `"I"`, `"love"`, and `"fastTextWordEmbedding"` are in the word embedding.

```
words = ["I" "love" "fastTextWordEmbedding"];
tf = isVocabularyWord(emb,words)
```

```
tf = 1×3 logical array

   1   1   0
```

Input Arguments

`emb` — Input word embedding
`wordEmbedding` object

Input word embedding, specified as a `wordEmbedding` object.

`enc` — Input word encoding
`wordEncoding` object

Input word encoding, specified as a `wordEncoding` object.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

See Also

`doc2sequence` | `fastTextWordEmbedding` | `tokenizedDocument` | `vec2word` | `word2vec` | `wordEmbedding`

Topics
"Train a Sentiment Classifier"
"Classify Text Data Using Deep Learning"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2018b

join

Combine multiple bag-of-words or bag-of-n-grams models

Syntax

```
newBag = join(bag)
newBag = join(bag,dim)
```

Description

`newBag = join(bag)` combines the elements in the array `bag` by merging the frequency counts. The function combines the elements along the first dimension not equal to 1.

`newBag = join(bag,dim)` combines the elements in the array `bag` along the dimension `dim`.

Examples

Combine Bag-of-Words Models

Create an array of two bags-of-words models from tokenized documents.

```
str = [ ...
    "an example of a short sentence"
    "a second short sentence"];
documents = tokenizedDocument(str);
bag(1) = bagOfWords(documents(1));
bag(2) = bagOfWords(documents(2))
```

```
bag=1×2 object
  1x2 bagOfWords array with properties:

    Counts
    Vocabulary
    NumWords
    NumDocuments
```

Combine the bag-of-words models using `join`.

```
bag = join(bag)
```

```
bag = 
  bagOfWords with properties:

          Counts: [2x7 double]
      Vocabulary: [1x7 string]
        NumWords: 7
    NumDocuments: 2
```

Create Bag-of-Words Model in Parallel

If your text data is contained in multiple files in a folder, then you can import the text data and create a bag-of-words model in parallel using `parfor`. If you have Parallel Computing Toolbox™ installed, then the `parfor` loop runs in parallel, otherwise, it runs in serial. Use `join` to combine an array of bag-of-words models into one model.

Create a bag-of-words model from a collection of files. The examples sonnets have file names "`exampleSonnetN.txt`", where `N` is the number of the sonnet. Get a list of the files and their locations using `dir`.

```
fileLocation = fullfile(matlabroot,'examples','textanalytics','exampleSonnet*.txt');
fileInfo = dir(fileLocation)
```

```
fileInfo = 

  0x1 empty struct array with fields:

    name
    folder
    date
    bytes
    isdir
    datenum
```

Initialize an empty bag-of-words model and then loop over the files and create an array of bag-of-words models.

```
bag = bagOfWords;

numFiles = numel(fileInfo);
parfor i = 1:numFiles
    f = fileInfo(i);
    filename = fullfile(f.folder,f.name);

    textData = extractFileText(filename);
    document = tokenizedDocument(textData);
    bag(i) = bagOfWords(document);
end
```

Combine the bag-of-words models using `join`.

```
bag = join(bag)
```

```
bag = 
  bagOfWords with properties:

          Counts: []
      Vocabulary: [1x0 string]
        NumWords: 0
    NumDocuments: 0
```

Input Arguments

bag — Array of bag-of-words or bag-of-n-grams models
`bagOfWords` array | `bagOfNgrams` array

Array of bag-of-words or bag-of-n-grams models, specified as a `bagOfWords` array or a `bagOfNgrams` array. If `bag` is a `bagOfNgrams` array, then each element to be joined must have the same value for the `NgramLengths` property.

`dim` — Dimension along which to join models
positive integer

Dimension along which to join models, specified as a positive integer. If `dim` is not specified, then the default is the first dimension with a size that does not equal 1.

Output Arguments

`newBag` — Output model
`bagOfWords` array | `bagOfNgrams` array

Output model, returned as a `bagOfWords` object or a `bagOfNgrams` object. The type of `newBag` is the same as the type of `bag`. `newBag` has the same data type as the input model and has a size of 1 along the dimension being joined.

See Also

`addDocument` | `bagOfNgrams` | `bagOfWords` | `encode` | `removeDocument` | `removeEmptyDocuments` | `tfidf` | `tokenizedDocument` | `topkngrams` | `topkwords`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2018a

joinWords

Convert documents to string by joining words

Syntax

```
newStr = joinWords(documents)
newStr = joinWords(documents,delim)
```

Description

`newStr = joinWords(documents)` converts a `tokenizedDocument` array to a string array by joining the words in each document with a space.

`newStr = joinWords(documents,delim)` joins the words with delimiter `delim` instead of a space.

Examples

Convert Documents to String by Joining Words

Convert a `tokenizedDocument` array to a string array by joining the words with a space.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"])
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

```
str = joinWords(documents)
```

```
str = 2x1 string
    "an example of a short sentence"
    "a second short sentence"
```

Convert a `tokenizedDocument` array to a string array by joining the words with an underscore.

```
str = joinWords(documents,"_")
```

```
str = 2x1 string
    "an_example_of_a_short_sentence"
    "a_second_short_sentence"
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`delim` — Delimiter to join words
string scalar | character vector | scalar cell array

Delimiter to join words, specified as a string scalar, character vector, or scalar cell array containing a character vector.

Example: `"_"`

Example: `'_'`

Example: `{'_'}`

Data Types: `char` | `string` | `cell`

Output Arguments

`newStr` — Output text
string array | character vector | cell array of character vectors

Output text, returned as a string array, a character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

See Also

`context` | `doc2cell` | `doclength` | `string` | `tokenizedDocument`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

knnsearch

Find nearest neighbors by edit distance

Syntax

```
idx = knnsearch(eds,words)
[idx,d] = knnsearch(eds,words)
[idx,d] = knnsearch(eds,words,Name,Value)
```

Description

`idx = knnsearch(eds,words)` finds the indices of the nearest neighbors in the edit distance searcher `eds` to each element in `words`.

`[idx,d] = knnsearch(eds,words)` also returns the edit distances between the elements of `words` and the nearest neighbors.

`[idx,d] = knnsearch(eds,words,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Find Nearest Words

Create an edit distance searcher.

```
vocabulary = ["Text" "Analytics" "Toolbox"];
eds = editDistanceSearcher(vocabulary,2);
```

Find the nearest words to `"Test"` and `"Analysis"`.

```
words = ["Test" "Analysis"];
idx = knnsearch(eds,words)
```

```
idx = 2×1

     1
     2
```

Get the words from the vocabulary using the returned indices.

```
nearestWords = eds.Vocabulary(idx)
```

```
nearestWords = 1x2 string
    "Text"    "Analytics"
```

Find Edit Distances to Nearest Words

Create an edit distance searcher.

```
vocabulary = ["MATLAB" "Text" "Analytics" "Toolbox"];
eds = editDistanceSearcher(vocabulary,2);
```

Find the nearest words and their edit distances to `"Test"` and `"Analysis"`.

```
words = ["Test" "Analysis"];
[idx,d] = knnsearch(eds,words)
```

```
idx = 2×1

     2
     3


d = 2×1

     1
     2
```

Get the words from the vocabulary using the returned indices.

```
nearestWords = eds.Vocabulary(idx)
```

```
nearestWords = 1x2 string
    "Text"    "Analytics"
```

Changing the word `"Test"` to `"Text"` requires one edit: a substitution. Changing the word `"Analysis"` into `"Analytics"` requires two edits: a substitution and an insertion.

Find Multiple Neighbors

Create an edit distance searcher.

```
vocabulary = ["MathWorks" "MATLAB" "Analytics"];
eds = editDistanceSearcher(vocabulary,5);
```

Find the two nearest words and their edit distances to `"Math"` and `"Analysis"`.

```
words = ["Math" "Analysis"];
idx = knnsearch(eds,words,'K',2)
```

```
idx = 2×2

     1     2
     3   NaN
```

View the two closest words to `"Math"`.

```
idxMath = idx(1,:);
newWords = eds.Vocabulary(idxMath)
newWords = 1x2 string
    "MathWorks"    "MATLAB"
```

There is only one word within the maximum edit distance from `"Analysis"`, so the function returns `NaN` for the other indices. View the nearest words with valid indices.

```
idxAnalysis = idx(2,:);
idxAnalysis(isnan(idxAnalysis)) = [];
newWords = eds.Vocabulary(idxAnalysis)
```

```
newWords = 
"Analytics"
```

Input Arguments

eds — Edit distance searcher
`editDistanceSearcher`

Edit distance searcher, specified as an `editDistanceSearcher` object.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `knnsearch(eds,words,'K',3)` finds the nearest three neighbors in `eds` to the elements of `words`.

`K` — Number of nearest neighbors to find
1 (default) | positive integer

Number of nearest neighbors to find for each element in `words`, specified as a positive integer.

Example: `'K',3`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`IncludeTies` — Option to include neighbors whose distance values are equal
`false` (default) | `true`

Option to return neighbors whose distance values are equal, specified as `true` or `false`.

If `'IncludeTies'` is `false`, then the function returns the *K* neighbors with the shortest edit distance, where *K* is the number of neighbors to find. In this case, the function outputs *N*-by-*K* matrices, where *N* is the number of input words. To specify *K*, use the `'K'` name-value pair argument.

If `'IncludeTies'` is `true`, then the function also returns the neighbors whose distances are equal to the *K*th smallest distance in the output. In this case, the function outputs cell arrays of size *N*-by-1, where *N* is the number of input words. The elements of the cell arrays are vectors with at least *K* elements. The function sorts the neighbors in each vector in ascending order of distance.

Example: `'IncludeTies',true`

Data Types: `logical`

Output Arguments

`idx` — Indices of nearest neighbors in searcher
matrix | cell array of vectors

Indices of nearest neighbors in the searcher, returned as a matrix or a cell array of vectors.

If `'IncludeTies'` is `false`, then the function returns the *K* neighbors with the shortest edit distance, where *K* is the number of neighbors to find. In this case, the function outputs *N*-by-*K* matrices, where *N* is the number of input words. To specify *K*, use the `'K'` name-value pair argument.

If `'IncludeTies'` is `true`, then the function also returns the neighbors whose distances are equal to the *K*th smallest distance in the output. In this case, the function outputs cell arrays of size *N*-by-1, where *N* is the number of input words. The elements of the cell arrays are vectors with at least *K* elements. The function sorts the neighbors in each vector in ascending order of distance.

Data Types: `double` | `cell`

`d` — Edit distances to neighbors
matrix | cell array of vectors

Edit distances to neighbors, returned as a matrix or a cell array of vectors.

If `'IncludeTies'` is `false`, then the function returns the *K* neighbors with the shortest edit distance, where *K* is the number of neighbors to find. In this case, the function outputs *N*-by-*K* matrices, where *N* is the number of input words. To specify *K*, use the `'K'` name-value pair argument.

If `'IncludeTies'` is `true`, then the function also returns the neighbors whose distances are equal to the *K*th smallest distance in the output. In this case, the function outputs cell arrays of size *N*-by-1, where *N* is the number of input words. The elements of the cell arrays are vectors with at least *K* elements. The function sorts the neighbors in each vector in ascending order of distance.

Data Types: `double` | `cell`

See Also

`correctSpelling` | `editDistance` | `editDistanceSearcher` | `rangesearch` | `splitGraphemes` | `tokenizedDocument`

Topics
"Correct Spelling in Documents"
"Create Extension Dictionary for Spelling Correction"
"Create Custom Spelling Correction Function Using Edit Distance Searchers"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"

Introduced in R2019a

ldaModel

Latent Dirichlet allocation (LDA) model

Description

A latent Dirichlet allocation (LDA) model is a topic model which discovers underlying topics in a collection of documents and infers word probabilities in topics. If the model was fit using a bag-of-n-grams model, then the software treats the n-grams as individual words.

Creation

Create an LDA model using the `fitlda` function.

Properties

`NumTopics` — Number of topics
positive integer

Number of topics in the LDA model, specified as a positive integer.

`TopicConcentration` — Topic concentration
positive scalar

Topic concentration, specified as a positive scalar. The function sets the concentration per topic to `TopicConcentration/NumTopics`. For more information, see “Latent Dirichlet Allocation” on page 1-214.

`WordConcentration` — Word concentration
`1` (default) | nonnegative scalar

Word concentration, specified as a nonnegative scalar. The software sets the concentration per word to `WordConcentration/numWords`, where `numWords` is the vocabulary size of the input documents. For more information, see “Latent Dirichlet Allocation” on page 1-214.

`CorpusTopicProbabilities` — Topic probabilities of input document set
vector

Topic probabilities of input document set, specified as a vector. The corpus topic probabilities of an LDA model are the probabilities of observing each topic in the entire data set used to fit the LDA model. `CorpusTopicProbabilities` is a 1-by-K vector where K is the number of topics. The kth entry of `CorpusTopicProbabilities` corresponds to the probability of observing topic k.

`DocumentTopicProbabilities` — Topic probabilities per input document
matrix

Topic probabilities per input document, specified as a matrix. The document topic probabilities of an LDA model are the probabilities of observing each topic in each document used to fit the LDA model. `DocumentTopicProbabilities` is a D-by-K matrix where D is the number of documents used to fit the LDA model, and *K* is the number of topics. The *(d,k)*th entry of `DocumentTopicProbabilities` corresponds to the probability of observing topic *k* in document *d*.

If any the topics have zero probability (`CorpusTopicProbabilities` contains zeros), then the corresponding columns of `DocumentTopicProbabilities` and `TopicWordProbabilities` are zeros.

The order of the rows in `DocumentTopicProbabilities` corresponds to the order of the documents in the training data.

`TopicWordProbabilities` — Word probabilities per topic
matrix

Word probabilities per topic, specified as a matrix. The topic word probabilities of an LDA model are the probabilities of observing each word in each topic of the LDA model. `TopicWordProbabilities` is a *V*-by-*K* matrix, where *V* is the number of words in `Vocabulary` and *K* is the number of topics. The *(v,k)*th entry of `TopicWordProbabilities` corresponds to the probability of observing word *v* in topic *k*.

If any the topics have zero probability (`CorpusTopicProbabilities` contains zeros), then the corresponding columns of `DocumentTopicProbabilities` and `TopicWordProbabilities` are zeros.

The order of the rows in `TopicWordProbabilities` corresponds to the order of the words in `Vocabulary`.

`TopicOrder` — Topic order
`'initial-fit-probability'` (default) | `'unordered'`

Topic order, specified as one of the following:

- `'initial-fit-probability'` – Sort the topics by the corpus topic probabilities of the initial model fit. These probabilities are the `CorpusTopicProbabilities` property of the initial `ldaModel` object returned by `fitlda`. The `resume` function does not reorder the topics of the resulting `ldaModel` objects.
- `'unordered'` – Do not order topics.

`FitInfo` — Information recorded when fitting LDA model
struct

Information recorded when fitting LDA model, specified as a struct with the following fields:

- `TerminationCode` – Status of optimization upon exit
  - 0 – Iteration limit reached.
  - 1 – Tolerance on log-likelihood satisfied.
- `TerminationStatus` – Explanation of the returned termination code
- `NumIterations` – Number of iterations performed
- `NegativeLogLikelihood` – Negative log-likelihood for the data passed to `fitlda`
- `Perplexity` – Perplexity for the data passed to `fitlda`
- `Solver` – Name of the solver used

- `History` – Struct holding the optimization history
- `StochasticInfo` – Struct holding information for stochastic solvers

Data Types: `struct`

`Vocabulary` — List of words in the model
string vector

List of words in the model, specified as a string vector.

Data Types: `string`

Object Functions

logp	Document log-probabilities and goodness of fit of LDA model
predict	Predict top LDA topics of documents
resume	Resume fitting LDA model
topkwords	Most important words in bag-of-words model or LDA topic
transform	Transform documents into lower-dimensional space
wordcloud	Create word cloud chart from text, bag-of-words model, bag-of-n-grams model, or LDA model

Examples

Fit LDA Model

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

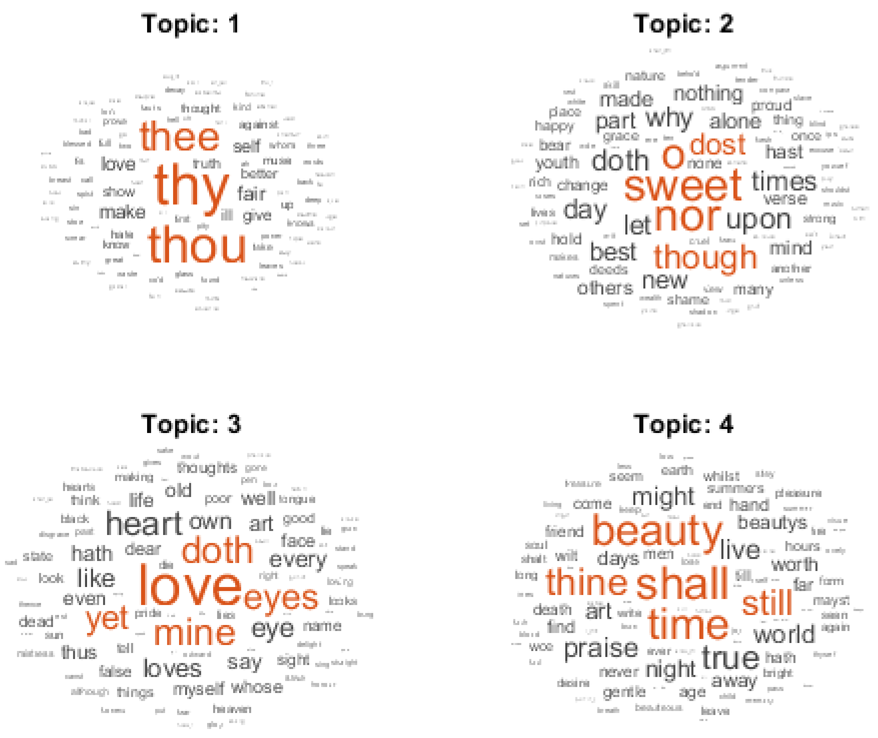
Fit an LDA model with four topics.

```
numTopics = 4;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0887969 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration | change in  | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.01 |            |  1.215e+03 |         1.000 |             0 |
|          1 |       0.02 | 1.0482e-02 |  1.128e+03 |         1.000 |             0 |
|          2 |       0.02 | 1.7190e-03 |  1.115e+03 |         1.000 |             0 |
|          3 |       0.02 | 4.3796e-04 |  1.118e+03 |         1.000 |             0 |
|          4 |       0.02 | 9.4193e-04 |  1.111e+03 |         1.000 |             0 |
|          5 |       0.02 | 3.7079e-04 |  1.108e+03 |         1.000 |             0 |
|          6 |       0.02 | 9.5777e-05 |  1.107e+03 |         1.000 |             0 |
=====================================================================================

mdl = 
  ldaModel with properties:

                     NumTopics: 4
             WordConcentration: 1
            TopicConcentration: 1
      CorpusTopicProbabilities: [0.2500 0.2500 0.2500 0.2500]
    DocumentTopicProbabilities: [154x4 double]
        TopicWordProbabilities: [3092x4 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Visualize the topics using word clouds.

```
figure
for topicIdx = 1:4
    subplot(2,2,topicIdx)
    wordcloud(mdl,topicIdx);
    title("Topic: " + topicIdx)
end
```



Highest Probability Words of LDA Topic

Create a table of the words with highest probability of an LDA topic.

To reproduce the results, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents);
```

Fit an LDA model with 20 topics. To suppress verbose output, set `'Verbose'` to 0.

```
numTopics = 20;
mdl = fitlda(bag,numTopics,'Verbose',0);
```

Find the top 20 words of the first topic.

```
k = 20;
topicIdx = 1;
tbl = topkwords(mdl,k,topicIdx)
```

```
tbl=20×2 table
      Word        Score  
    ________    _________

    "eyes"        0.11155
    "beauty"      0.05777
    "hath"       0.055778
    "still"      0.049801
    "true"       0.043825
    "mine"       0.033865
    "find"       0.031873
    "black"      0.025897
    "look"       0.023905
    "tis"        0.023905
    "kind"       0.021913
    "seen"       0.021913
    "found"      0.017929
    "sin"        0.015937
    "three"      0.013945
    "golden"    0.0099608
       ⋮
```

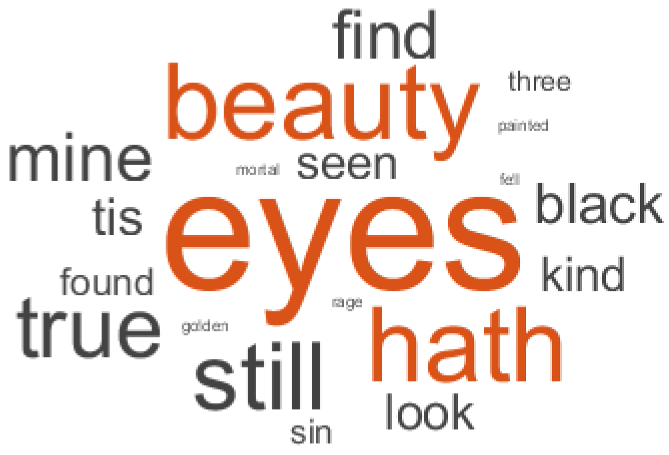
Find the top 20 words of the first topic and use inverse mean scaling on the scores.

```
tbl = topkwords(mdl,k,topicIdx,'Scaling','inversemean')
```

```
tbl=20×2 table
      Word       Score  
    ________    ________

    "eyes"        1.2718
    "beauty"     0.59022
    "hath"        0.5692
    "still"      0.50269
    "true"       0.43719
    "mine"       0.32764
    "find"       0.32544
    "black"      0.25931
    "tis"        0.23755
    "look"       0.22519
    "kind"       0.21594
    "seen"       0.21594
    "found"      0.17326
    "sin"        0.15223
    "three"      0.13143
    "golden"    0.090698
       ⋮
```

Create a word cloud using the scaled scores as the size data.

```
figure
wordcloud(tbl.Word,tbl.Score);
```



Document Topic Probabilities of LDA Model

Get the document topic probabilities (also known as topic mixtures) of the documents used to fit an LDA model.

To reproduce the results, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents);
```

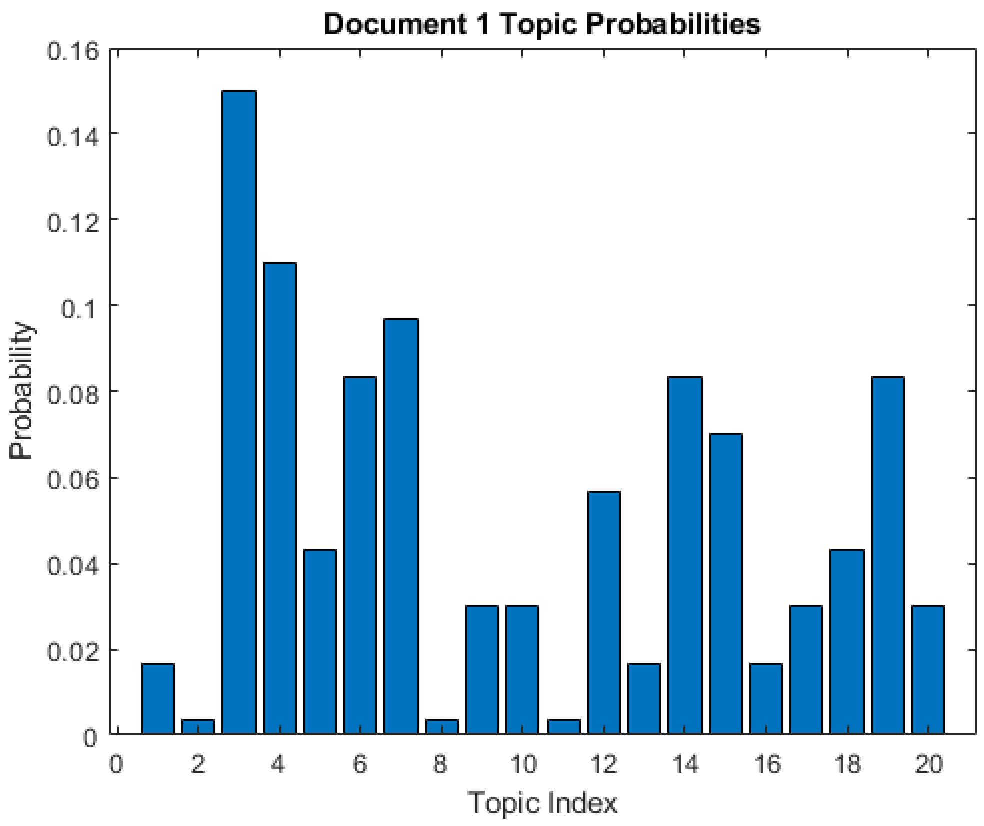
Fit an LDA model with 20 topics. To suppress verbose output, set `'Verbose'` to 0.

```
numTopics = 20;
mdl = fitlda(bag,numTopics,'Verbose',0)
```

```
mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

View the topic probabilities of the first document in the training data.

```
topicMixtures = mdl.DocumentTopicProbabilities;
figure
bar(topicMixtures(1,:))
title("Document 1 Topic Probabilities")
xlabel("Topic Index")
ylabel("Probability")
```



Predict Top LDA Topics of Documents

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LDA model with 20 topics.

```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0612641 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration | change in  | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.26 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.06 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.06 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.06 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.06 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.06 | 6.4180e-05 |  7.291e+02 |         5.000 |             0 |
=====================================================================================

mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
```

```
          TopicOrder: 'initial-fit-probability'
             FitInfo: [1x1 struct]
```

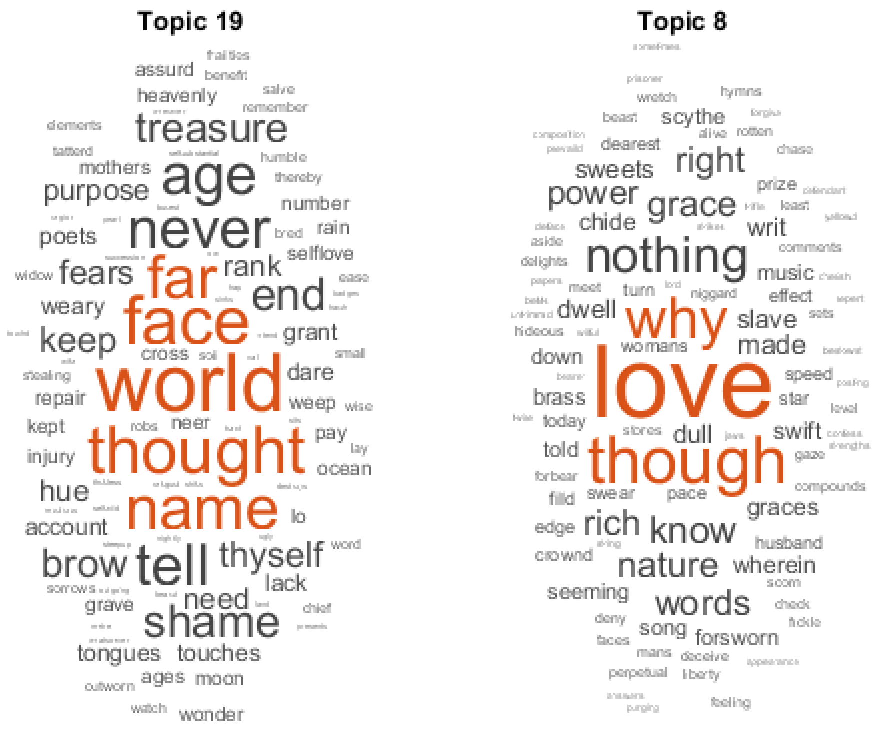
Predict the top topics for an array of new documents.

```
newDocuments = tokenizedDocument([
    "what's in a name? a rose by any other name would smell as sweet."
    "if music be the food of love, play on."]);
topicIdx = predict(mdl,newDocuments)
```

```
topicIdx = 2×1

    19
     8
```

Visualize the predicted topics using word clouds.

```
figure
subplot(1,2,1)
wordcloud(mdl,topicIdx(1));
title("Topic " + topicIdx(1))
subplot(1,2,2)
wordcloud(mdl,topicIdx(2));
title("Topic " + topicIdx(2))
```



More About

Latent Dirichlet Allocation

A *latent Dirichlet allocation* (LDA) model is a document topic model which discovers underlying topics in a collection of documents and infers word probabilities in topics. LDA models a collection of D documents as topic mixtures $\theta_1, \ldots, \theta_D$, over K topics characterized by vectors of word probabilities $\varphi_1, \ldots, \varphi_K$. The model assumes that the topic mixtures $\theta_1, \ldots, \theta_D$, and the topics $\varphi_1, \ldots, \varphi_K$ follow a Dirichlet distribution with concentration parameters α and β respectively.

The topic mixtures $\theta_1, \ldots, \theta_D$ are probability vectors of length K, where K is the number of topics. The entry θ_{di} is the probability of topic i appearing in the dth document. The topic mixtures correspond to the rows of the `DocumentTopicProbabilities` property of the `ldaModel` object.

The topics $\varphi_1, \ldots, \varphi_K$ are probability vectors of length V, where V is the number of words in the vocabulary. The entry φ_{iv} corresponds to the probability of the vth word of the vocabulary appearing in the ith topic. The topics $\varphi_1, \ldots, \varphi_K$ correspond to the columns of the `TopicWordProbabilities` property of the `ldaModel` object.

Given the topics $\varphi_1, \ldots, \varphi_K$ and Dirichlet prior α on the topic mixtures, LDA assumes the following generative process for a document:

1. Sample a topic mixture $\theta \sim \mathrm{Dirichlet}(\alpha)$. The random variable θ is a probability vector of length K, where K is the number of topics.
2. For each word in the document:
   - a. Sample a topic index $z \sim \mathrm{Categorical}(\theta)$. The random variable z is an integer from 1 through K, where K is the number of topics.
   - b. Sample a word $w \sim \mathrm{Categorical}(\varphi_z)$. The random variable w is an integer from 1 through V, where V is the number of words in the vocabulary, and represents the corresponding word in the vocabulary.

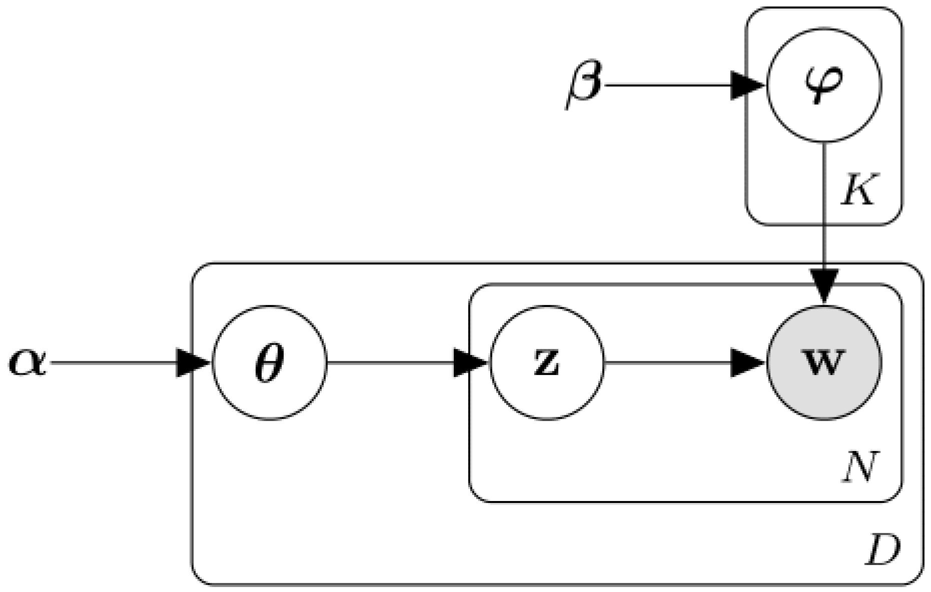
Under this generative process, the joint distribution of a document with words $w_1, \ldots, w_N$, with topic mixture θ, and with topic indices $z_1, \ldots, z_N$ is given by

$$p(\theta, z, w \,\middle|\, \alpha, \varphi) = p(\theta \,\middle|\, \alpha) \prod_{n=1}^{N} p(z_n \,\middle|\, \theta) p(w_n \,\middle|\, z_n, \varphi),$$

where N is the number of words in the document. Summing the joint distribution over z and then integrating over θ yields the marginal distribution of a document w:

$$p(w \,\middle|\, \alpha, \varphi) = \int_{\theta} p(\theta \,\middle|\, \alpha) \prod_{n=1}^{N} \sum_{z_n} p(z_n | \theta) p(w_n | z_n, \varphi) d\theta\,.$$

The following diagram illustrates the LDA model as a probabilistic graphical model. Shaded nodes are observed variables, unshaded nodes are latent variables, nodes without outlines are the model parameters. The arrows highlight dependencies between random variables and the plates indicate repeated nodes.



Dirichlet Distribution

The *Dirichlet distribution* is a continuous generalization of the multinomial distribution. Given the number of categories $K \geq 2$, and concentration parameter α, where α is a vector of positive reals of length K, the probability density function of the Dirichlet distribution is given by

$$p(\theta \mid \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} \theta_i^{\alpha_i - 1},$$

where B denotes the multivariate Beta function given by

$$B(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}.$$

A special case of the Dirichlet distribution is the *symmetric Dirichlet distribution*. The symmetric Dirichlet distribution is characterized by the concentration parameter α, where all the elements of α are the same.

See Also

`bagOfWords` | `fitlda` | `logp` | `lsaModel` | `predict` | `resume` | `topkwords` | `transform` | `wordcloud`

Topics

“Analyze Text Data Using Topic Models”
“Choose Number of Topics for LDA Model”
“Compare LDA Solvers”
“Analyze Text Data Using Multiword Phrases”
“Classify Text Data Using Deep Learning”

Introduced in R2017b

lexrankScores

Document scoring with LexRank algorithm

Syntax

```
scores = lexrankScores(documents)
scores = lexrankScores(bag)
```

Description

`scores = lexrankScores(documents)` scores the specified documents for importance according to pairwise similarity values using the LexRank algorithm. The function uses cosine similarity, and computes importance using the PageRank algorithm.

`scores = lexrankScores(bag)` scores documents encoded by a bag-of-words or bag-of-n-grams model.

Examples

Importance of Documents

Create an array of tokenized documents.

```
str = [
    "the quick brown fox jumped over the lazy dog"
    "the fast brown fox jumped over the lazy dog"
    "the lazy dog sat there and did nothing"
    "the other animals sat there watching"];
documents = tokenizedDocument(str)
```

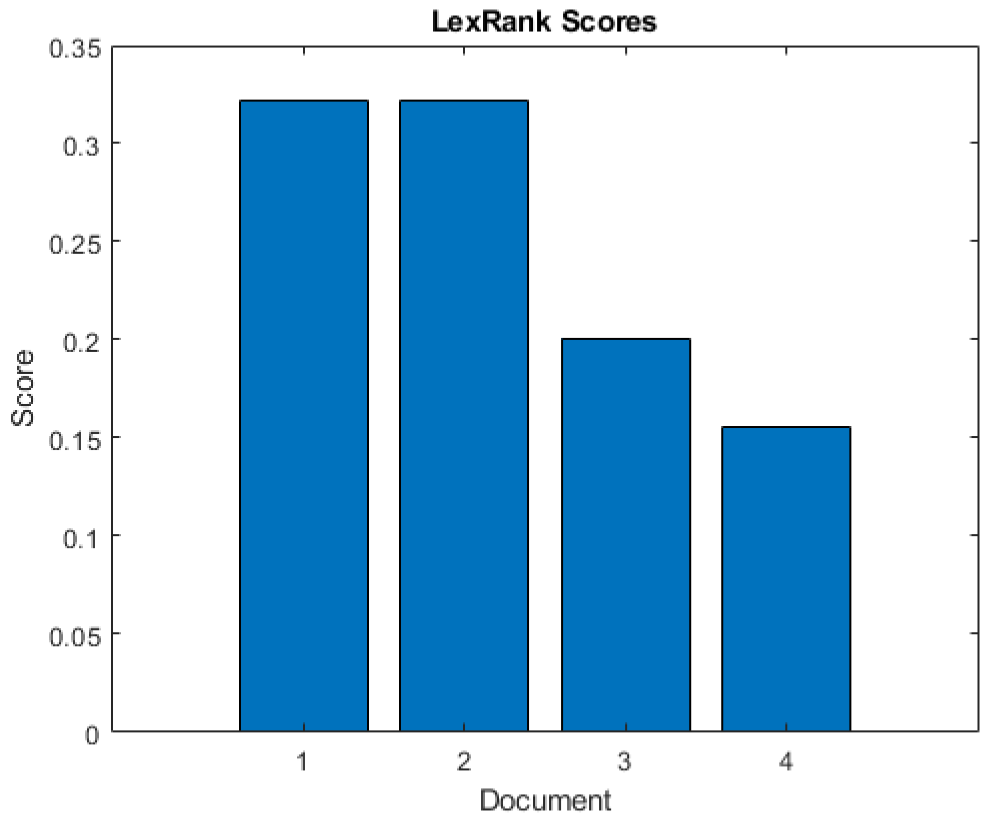
```
documents = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    9 tokens: the fast brown fox jumped over the lazy dog
    8 tokens: the lazy dog sat there and did nothing
    6 tokens: the other animals sat there watching
```

Calculate their LexRank scores.

```
scores = lexrankScores(documents);
```

Visualize the scores in a bar chart.

```
figure
bar(scores)
xlabel("Document")
ylabel("Score")
title("LexRank Scores")
```



Scores Using Bag-of-Words Model

Create a bag-of-words model from the text data in `sonnets.csv`.

```
filename = "sonnets.csv";
tbl = readtable(filename,'TextType','string');
textData = tbl.Sonnet;
documents = tokenizedDocument(textData);
bag = bagOfWords(documents)
```

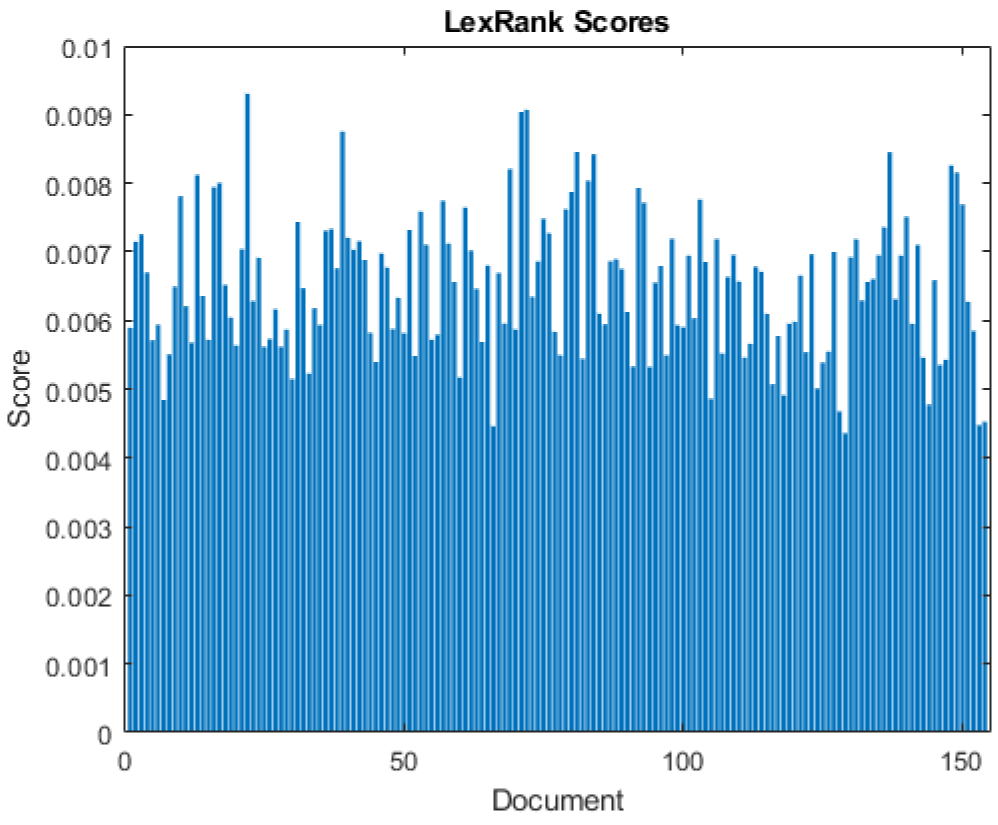
```
bag = 
  bagOfWords with properties:

          Counts: [154x3527 double]
      Vocabulary: [1x3527 string]
        NumWords: 3527
    NumDocuments: 154
```

Calculate LexRank scores for each sonnet.

```
scores = lexrankScores(bag);
```

Visualize the scores in a bar chart.

```
figure
bar(scores)
xlabel("Document")
ylabel("Score")
title("LexRank Scores")
```



Input Arguments

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

`bag` — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

Output Arguments

`scores` — LexRank scores
vector

LexRank scores, returned as a *N*-by-1 vector, where `scores(i)` corresponds to the score for the `i`th input document and *N* is the number of input documents.

References

[1] Erkan, Günes, and Dragomir R. Radev. "Lexrank: Graph-based Lexical Centrality as Salience in Text Summarization." *Journal of Artificial Intelligence Research 22* (2004): 457-479.

See Also

`bleuEvaluationScore` | `bm25Similarity` | `cosineSimilarity` | `extractSummary` | `mmrScores` | `rougeEvaluationScore` | `textrankScores` | `tokenizedDocument`

Topics
“Sequence-to-Sequence Translation Using Attention”

Introduced in R2020a

lsaModel

Latent semantic analysis (LSA) model

Description

A latent semantic analysis (LSA) model discovers relationships between documents and the words that they contain. An LSA model is a dimensionality reduction tool useful for running low-dimensional statistical models on high-dimensional word counts. If the model was fit using a bag-of-n-grams model, then the software treats the n-grams as individual words.

Creation

Create an LSA model using the `fitlsa` function.

Properties

`NumComponents` — Number of components
nonnegative integer

Number of components, specified as a nonnegative integer. The number of components is the dimensionality of the result vectors. Changing the value of `NumComponents` changes the length of the resulting vectors, without influencing the initial values. You can only set `NumComponents` to be less than or equal to the number of components used to fit the LSA model.

Example: 100

`FeatureStrengthExponent` — Exponent scaling feature component strengths
nonnegative scalar

Exponent scaling feature component strengths for the `DocumentScores` and `WordScores` properties, and the `transform` function, specified as a nonnegative scalar. The LSA model scales the properties by their singular values (feature strengths), with an exponent of `FeatureStrengthExponent/2`.

Example: 2.5

`ComponentWeights` — Component weights
numeric vector

Component weights, specified as a numeric vector. The component weights of an LSA model are the singular values, squared. `ComponentWeights` is a 1-by-`NumComponents` vector where the *j*th entry corresponds to the weight of component *j*. The components are ordered by decreasing weights. You can use the weights to estimate the importance of components.

`DocumentScores` — Score vectors per input document
matrix

Score vectors per input document, specified as a matrix. The document scores of an LSA model are the score vectors in lower dimensional space of each document used to fit the LSA model.

`DocumentScores` is a *D*-by-`NumComponents` matrix where *D* is the number of documents used to fit the LSA model. The *(i,j)*th entry of `DocumentScores` corresponds to the score of component *j* in document *i*.

`WordScores` — Word scores per component
matrix

Word scores per component, specified as a matrix. The word scores of an LSA model are the scores of each word in each component of the LSA model. `WordScores` is a *V*-by-`NumComponents` matrix where *V* is the number of words in `Vocabulary`. The *(v,j)*th entry of `WordScores` corresponds to the score of word *v* in component *j*.

`Vocabulary` — Unique words in model
string vector

Unique words in the model, specified as a string vector.

Data Types: `string`

Object Functions

transform	Transform documents into lower-dimensional space

Examples

Fit LSA Model

Fit a Latent Semantic Analysis model to a collection of documents.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LSA model with 20 components.

```
numComponents = 20;
mdl = fitlsa(bag,numComponents)
mdl = 
  lsaModel with properties:

              NumComponents: 20
           ComponentWeights: [1x20 double]
             DocumentScores: [154x20 double]
                 WordScores: [3092x20 double]
                 Vocabulary: [1x3092 string]
    FeatureStrengthExponent: 2
```

Transform new documents into lower dimensional space using the LSA model.

```
newDocuments = tokenizedDocument([
    "what's in a name? a rose by any other name would smell as sweet."
    "if music be the food of love, play on."]);
dscores = transform(mdl,newDocuments)
```

```
dscores = 2×20

    0.1338    0.1623    0.1680   -0.0541   -0.2464   -0.0134    0.2604   -0.0205   -0.1127    0.0
    0.2547    0.5576   -0.0095    0.5660   -0.0643   -0.1236   -0.0082    0.0522    0.0690   -0.0
```

Calculate Document Similarity

Create a bag-of-words model from some text data.

```
str = [
    "I enjoy ham, eggs and bacon for breakfast."
    "I sometimes skip breakfast."
    "I eat eggs and ham for dinner."
    ];
documents = tokenizedDocument(str);
bag = bagOfWords(documents);
```

Fit an LSA model with two components. Set the feature strength exponent to 0.5.

```
numComponents = 2;
exponent = 0.5;
mdl = fitlsa(bag,numComponents, ...
    'FeatureStrengthExponent',exponent)
```

```
mdl = 
  lsaModel with properties:

                NumComponents: 2
             ComponentWeights: [16.2268 4.0000]
               DocumentScores: [3x2 double]
                   WordScores: [14x2 double]
                   Vocabulary: [1x14 string]
    FeatureStrengthExponent: 0.5000
```

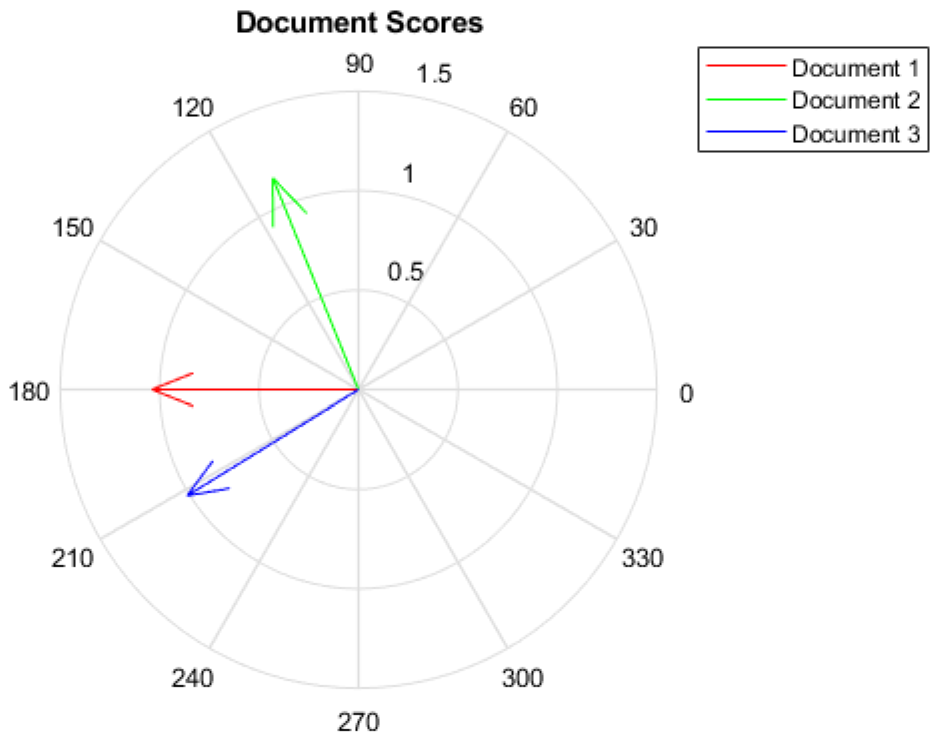
Calculate the cosine distance between the documents score vectors using `pdist`. View the distances in a matrix `D` using `squareform`. `D(i,j)` denotes the distance between document `i` and `j`.

```
dscores = mdl.DocumentScores;
distances = pdist(dscores,'cosine');
D = squareform(distances)
```

```
D = 3×3

         0    0.6244    0.1489
    0.6244         0    1.1670
    0.1489    1.1670         0
```

Visualize the similarity between documents by plotting the document score vectors in a compass plot.

```
figure
compass(dscores(1,1),dscores(1,2),'red')
hold on
compass(dscores(2,1),dscores(2,2),'green')
compass(dscores(3,1),dscores(3,2),'blue')
hold off
title("Document Scores")
legend(["Document 1" "Document 2" "Document 3"],'Location','bestoutside')
```



See Also

bagOfWords | fitlsa | ldaModel | lsaModel | transform

Topics

"Analyze Text Data Using Topic Models"

“Choose Number of Topics for LDA Model”
“Compare LDA Solvers”
“Analyze Text Data Using Multiword Phrases”
“Classify Text Data Using Deep Learning”

Introduced in R2017b

logp

Document log-probabilities and goodness of fit of LDA model

Syntax

```
logProb = logp(ldaMdl,documents)
logProb = logp(ldaMdl,counts)
logProb = logp(ldaMdl,bag)
[logProb,ppl] = logp( ___ )
 ___ = logp( ___ ,Name,Value)
```

Description

`logProb = logp(ldaMdl,documents)` returns the log-probabilities of `documents` under the LDA model `ldaMdl`.

`logProb = logp(ldaMdl,counts)` returns the log-probabilities of the documents represented by the matrix of word counts `counts`.

`logProb = logp(ldaMdl,bag)` returns the log-probabilities of the documents represented by a bag-of-words or bag-of-n-grams model.

`[logProb,ppl] = logp( ___ )` returns the perplexity computed from the log-probabilities.

`___ = logp( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Calculate Document Log-Probabilities

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

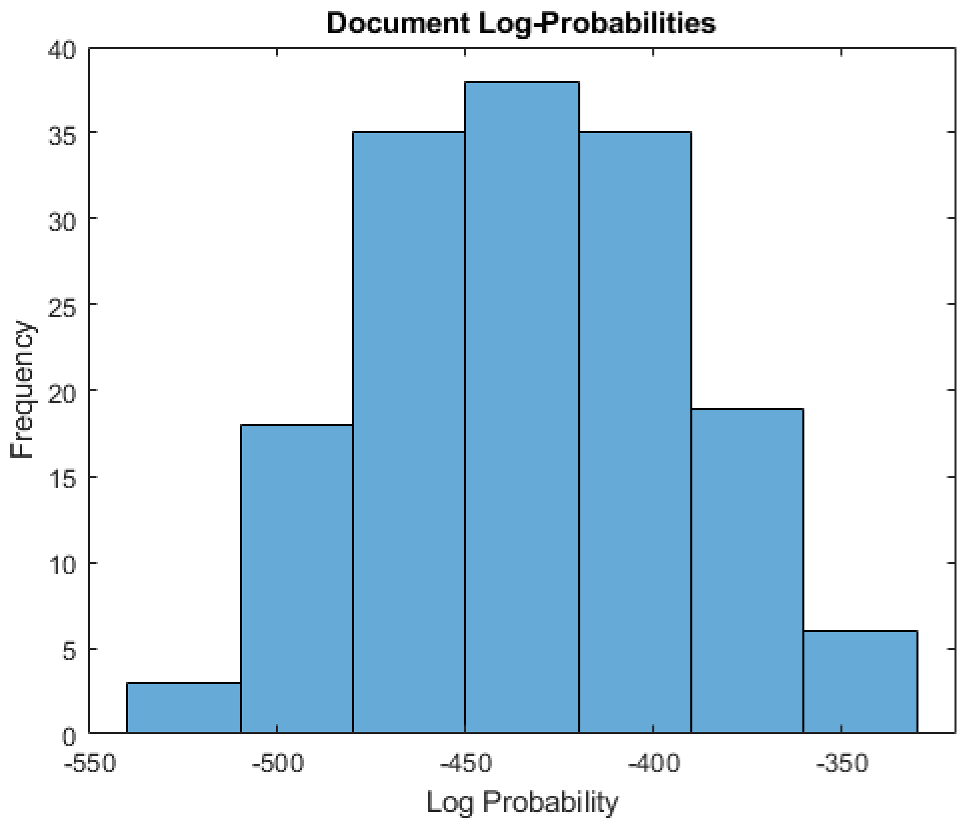
```
bag = 
  bagOfWords with properties:
          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LDA model with 20 topics. To suppress verbose output, set `'Verbose'` to 0.

```
numTopics = 20;
mdl = fitlda(bag,numTopics,'Verbose',0);
```

Compute the document log-probabilities of the training documents and show them in a histogram.

```
logProbabilities = logp(mdl,documents);
figure
histogram(logProbabilities)
xlabel("Log Probability")
ylabel("Frequency")
title("Document Log-Probabilities")
```



Identify the three documents with the lowest log-probability. A low log-probability may suggest that the document may be an outlier.

```
[~,idx] = sort(logProbabilities);
idx(1:3)
```

```
ans = 3×1
```

```
   146
    19
    65
```

```
documents(idx(1:3))
```

```
ans = 
  3x1 tokenizedDocument:

    76 tokens: poor soul centre sinful earth sinful earth rebel powers array why dost thou pine s
    76 tokens: devouring time blunt thou lions paws make earth devour own sweet brood pluck keen 
    73 tokens: brass nor stone nor earth nor boundless sea sad mortality oersways power rage sha
```

Calculate Document Log-Probabilities from Word Count Matrix

Load the example data. `sonnetsCounts.mat` contains a matrix of word counts and a corresponding vocabulary of preprocessed versions of Shakespeare's sonnets.

```
load sonnetsCounts.mat
size(counts)
```

```
ans = 1×2

         154        3092
```

Fit an LDA model with 20 topics.

```
numTopics = 20;
mdl = fitlda(counts,numTopics)
```

```
Initial topic assignments sampled in 0.0479481 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.07 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.06 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.06 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.06 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.07 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.06 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.06 | 6.4180e-05 |  7.291e+02 |         5.000 |             0 |
=====================================================================================
```

```
mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
```

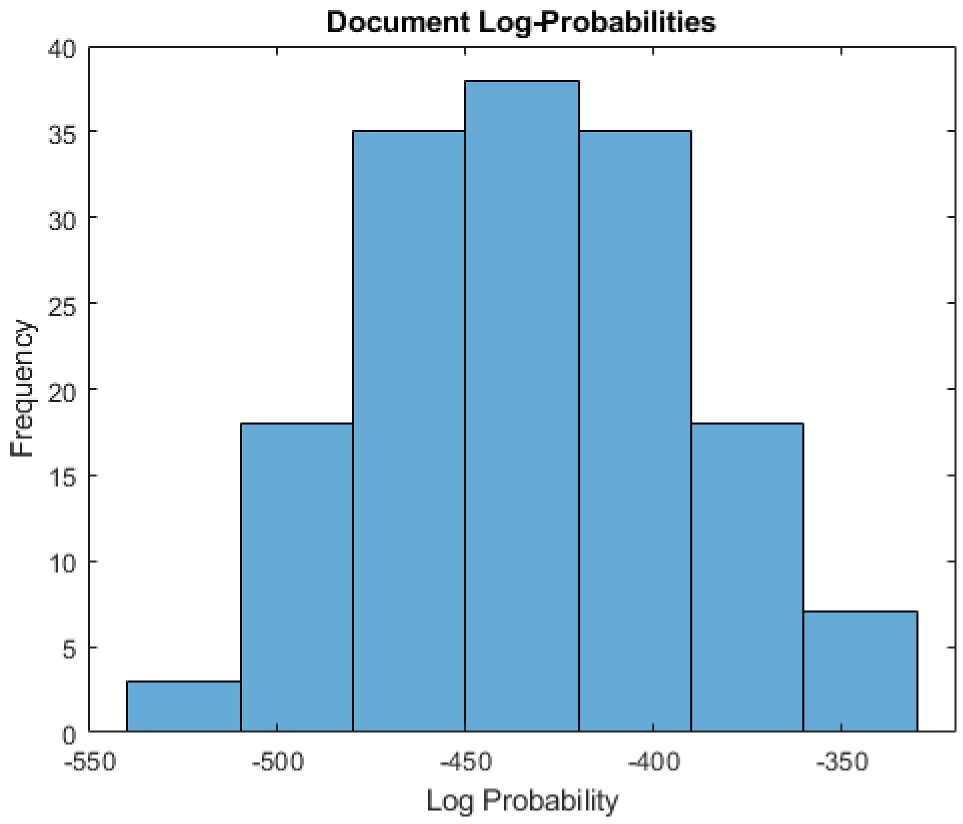
```
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Compute the document log-probabilities of the training documents. Specify to draw 500 samples for each document.

```
numSamples = 500;
logProbabilities = logp(mdl,counts, ...
    'NumSamples',numSamples);
```

Show the document log-probabilities in a histogram.

```
figure
histogram(logProbabilities)
xlabel("Log Probability")
ylabel("Frequency")
title("Document Log-Probabilities")
```



Identify the indices of the three documents with the lowest log-probability.

```
[~,idx] = sort(logProbabilities);
idx(1:3)
```

```
ans = 3×1
```

```
   146
    19
    65
```

Compare Goodness of Fit

Compare the goodness of fit for two LDA models by calculating the perplexity of a held-out test set of documents.

To reproduce the results, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Set aside 10% of the documents at random for testing.

```
numDocuments = numel(documents);
cvp = cvpartition(numDocuments,'HoldOut',0.1);
documentsTrain = documents(cvp.training);
documentsTest = documents(cvp.test);
```

Create a bag-of-words model from the training documents.

```
bag = bagOfWords(documentsTrain)
```

```
bag = 
  bagOfWords with properties:

          Counts: [139x2909 double]
      Vocabulary: [1x2909 string]
        NumWords: 2909
    NumDocuments: 139
```

Fit an LDA model with 20 topics to the bag-of-words model. To suppress verbose output, set `'Verbose'` to 0.

```
numTopics = 20;
mdl1 = fitlda(bag,numTopics,'Verbose',0);
```

View information about the model fit.

```
mdl1.FitInfo
```

```
ans = struct with fields:
          TerminationCode: 1
```

```
        TerminationStatus: "Relative tolerance on log-likelihood satisfied."
            NumIterations: 26
    NegativeLogLikelihood: 5.6915e+04
               Perplexity: 742.7118
                   Solver: "cgs"
                  History: [1x1 struct]
```

Compute the perplexity of the held-out test set.

```
[~,ppl1] = logp(mdl1,documentsTest)
```

```
ppl1 = 781.6078
```

Fit an LDA model with 40 topics to the bag-of-words model.

```
numTopics = 40;
mdl2 = fitlda(bag,numTopics,'Verbose',0);
```

View information about the model fit.

```
mdl2.FitInfo
```

```
ans = struct with fields:
          TerminationCode: 1
        TerminationStatus: "Relative tolerance on log-likelihood satisfied."
            NumIterations: 37
    NegativeLogLikelihood: 5.4466e+04
               Perplexity: 558.8685
                   Solver: "cgs"
                  History: [1x1 struct]
```

Compute the perplexity of the held-out test set.

```
[~,ppl2] = logp(mdl2,documentsTest)
```

```
ppl2 = 808.6602
```

A lower perplexity suggests that the model may be better fit to the held-out test data.

Input Arguments

`ldaMdl` — Input LDA model
`ldaModel` object

Input LDA model, specified as an `ldaModel` object.

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

bag — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`counts` — Frequency counts of words
matrix of nonnegative integers

Frequency counts of words, specified as a matrix of nonnegative integers. If you specify `'DocumentsIn'` to be `'rows'`, then the value `counts(i,j)` corresponds to the number of times the *j*th word of the vocabulary appears in the *i*th document. Otherwise, the value `counts(i,j)` corresponds to the number of times the *i*th word of the vocabulary appears in the *j*th document.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NumSamples',500` specifies to draw 500 samples for each document

`DocumentsIn` — Orientation of documents
`'rows'` (default) | `'columns'`

Orientation of documents in the word count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Input is a matrix of word counts with rows corresponding to documents.
- `'columns'` – Input is a transposed matrix of word counts with columns corresponding to documents.

This option only applies if you specify the input documents as a matrix of word counts.

Note If you orient your word count matrix so that documents correspond to columns and specify `'DocumentsIn','columns'`, then you might experience a significant reduction in optimization-execution time.

`NumSamples` — Number of samples to draw
1000 (default) | positive integer

Number of samples to draw for each document, specified as the comma-separated pair consisting of `'NumSamples'` and a positive integer.

Example: `'NumSamples',500`

Output Arguments

`logProb` — Log-probabilities
numeric vector

Log-probabilities of the documents under the LDA model, returned as a numeric vector.

`ppl` — Perplexity
positive scalar

Perplexity of the documents calculated from the log-probabilities, returned as a positive scalar.

Algorithms

The `logp` uses the *iterated pseudo-count* method described in

References

[1] Wallach, Hanna M., Iain Murray, Ruslan Salakhutdinov, and David Mimno. "Evaluation methods for topic models." In *Proceedings of the 26th annual international conference on machine learning*, pp. 1105–1112. ACM, 2009. Harvard

See Also

`bagOfWords` | `fitlda` | `ldaModel` | `predict` | `transform` | `wordcloud`

Topics
“Analyze Text Data Using Topic Models”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

lower

Convert documents to lowercase

Syntax

`newDocuments = lower(documents)`

Description

`newDocuments = lower(documents)` converts each uppercase character in the input documents to the corresponding lowercase character, and leaves all other characters unchanged.

Examples

Convert Documents to Lowercase

Convert all uppercase characters in an array of documents to lowercase.

```
documents = tokenizedDocument([
    "An Example of a Short Sentence"
    "A Second Short Sentence"])
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: An Example of a Short Sentence
    4 tokens: A Second Short Sentence
```

```
newDocuments = lower(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

See Also
`decodeHTMLEntities` | `erasePunctuation` | `eraseTags` | `eraseURLs` | `tokenizedDocument` | `upper`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

mecabOptions

Options for MeCab tokenization

Description

A `mecabOptions` object specifies additional options for tokenizing Japanese and Korean text.

To tokenize using the specified MeCab tokenization options, use the `'TokenizeMethod'` option of `tokenizedDocument`.

Creation

Syntax

```
options = mecabOptions
options = mecabOptions(Name,Value)
```

Description

`options = mecabOptions` creates a MeCab tokenization option set with the default values for tokenizing Japanese.

`options = mecabOptions(Name,Value)` additionally sets additional "Properties" on page 1-236 using one or more name-value pair arguments.

Properties

`Model` — Path to trained model
string scalar | character vector

Path to trained model (MeCab dictionary), specified as a string scalar or a character vector.

The default value is a path to the internal dictionary for Japanese tokenization.

Example: `"C:\myDict"`

Data Types: `char` | `string`

`UserModel` — File containing model extension
`""` (default) | string scalar | character vector

File containing model extension (MeCab user dictionary `.dic` file), specified as a string scalar or a character vector.

Example: `"C:\myFile.dic"`

Data Types: `char` | `string`

`LemmaExtractor` — Function extracting lemma from MeCab reply
`@textanalytics.ja.mecabToLemma` (default) | function handle

Function extracting lemma from MeCab reply, specified as a function handle.

The function must have the form `lemmata = fun(words,info)`, where `words` is a string vector of tokens and `info` is a struct with the following fields:

- `Feature` – String vector of tokens of the same size as `words` containing the MeCab output lines in ChaSen format without the split tokens themselves.
- `PartOfSpeech` – Numerical code used inside the dictionary for the part-of-speech classification.

The output `lemmata` is a string array of the same size as `words` containing the extracted lemmata.

The default lemma extractor is the `textanalytics.ja.mecabToLemma` function.

Data Types: `function_handle`

`POSExtractor` — Function extracting part-of-speech information from MeCab reply
`@textanalytics.ja.mecabToPOS` (default) | function handle

Function extracting part-of-speech information from MeCab reply, specified as a function handle.

The function must have the form `posTags = fun(words,info)`, where `words` is a string vector of tokens and `info` is a struct with the following fields:

- `Feature` – String vector of tokens of the same size as `words` containing the MeCab output lines in ChaSen format without the split tokens themselves.
- `PartOfSpeech` – Numerical code used inside the dictionary for the part-of-speech classification.

The output `posTags` is a categorical array of the same size as `words` containing the extracted part-of-speech tags from the following categories:

- `adjective`
- `adposition`
- `adverb`
- `auxiliary-verb`
- `coord-conjunction`
- `determiner`
- `interjection`
- `noun`
- `numeral`
- `pronoun`
- `proper-noun`
- `punctuation`
- `symbol`
- `verb`
- `other`

The default part-of-speech information extractor is the `textanalytics.ja.mecabToPOS` function.

Data Types: `function_handle`

NERExtractor — Function extracting named entity information from MeCab reply
@textanalytics.ja.mecabToNER (default) | function handle

Function extracting named entity information from MeCab reply, specified as a function handle.

The function must have the form `entities = fun(words,info)`, where `words` is a string vector of tokens and `info` is a struct with the following fields:

- `Feature` – String vector of tokens of the same size as `words` containing the MeCab output lines in ChaSen format without the split tokens themselves.
- `PartOfSpeech` – Numerical code used inside the dictionary for the part-of-speech classification.

The output `entities` is a categorical array of the same size as `words` containing the extracted entities from the following categories:

- `non-entity`
- `person`
- `organization`
- `location`
- `other`

The default part-of-speech information extractor is the `textanalytics.ja.mecabToNER` function.

Data Types: `function_handle`

Examples

Create MeCab Options Object

Create a `mecabOptions` object containing the default options for Japanese tokenization.

```
options = mecabOptions
```

```
options = 

  MecabOptions with properties:

             Model: "C:\Program Files\MATLAB\R2019b\sys\share\dict-ipadic"
         UserModel: ""
    LemmaExtractor: @textanalytics.ja.mecabToLemma
      POSExtractor: @textanalytics.ja.mecabToPOS
      NERExtractor: @textanalytics.ja.mecabToNER
```

Specify MeCab User Dictionary for Tokenization

Tokenize Japanese text using custom MeCab options.

Create a string array of Japanese text.

```
str = [
    "恋に悩み、苦しむ。"
    "恋の悩みで苦しむ。"
    "空に星が輝き、瞬いている。"
    "空の星が輝きを増している。"];
```

Create a `mecabOptions` object and specify a user model as a `.dic` file using the `'UserModel'` option.

```
options = mecabOptions('UserModel','myFile.dic')
```

```
options = 

  MecabOptions with properties:

              Model: "C:\Program Files\MATLAB\R2019b\sys\share\dict-ipadic"
          UserModel: "myFile.dic"
     LemmaExtractor: @textanalytics.ja.mecabToLemma
       POSExtractor: @textanalytics.ja.mecabToPOS
       NERExtractor: @textanalytics.ja.mecabToNER
```

Tokenize the text using the specified options using the `'TokenizeMethod'` option.

```
documents = tokenizedDocument(str,'TokenizeMethod',options)
```

```
documents = 

  4×1 tokenizedDocument:

     6 tokens: 恋 に 悩み 、 苦しむ 。
     6 tokens: 恋 の 悩み で 苦しむ 。
    10 tokens: 空 に 星 が 輝き 、 瞬い て いる 。
    10 tokens: 空 の 星 が 輝き を 増し て いる 。
```

See Also
`addEntityDetails` | `addLanguageDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `corpusLanguage` | `normalizeWords` | `tokenDetails` | `tokenizedDocument`

Topics
“Japanese Language Support”
“Analyze Japanese Text Data”
“Language Considerations”
“Language-Independent Features”

Introduced in R2019b

mmrScores

Document scoring with Maximal Marginal Relevance (MMR) algorithm

Syntax

```
scores = mmrScores(documents,queries)
scores = mmrScores(bag,queries)
scores = mmrScores( ___ ,lambda)
```

Description

`scores = mmrScores(documents,queries)` scores `documents` according to their relevance to a `queries` avoiding redundancy using the MMR algorithm. The score in `scores(i,j)` is the MMR score of `documents(i)` relative to `queries(j)`.

`scores = mmrScores(bag,queries)` scores documents encoded by the bag-of-words or bag-of-n-grams model `bag` relative to `queries`. The score in `scores(i,j)` is the MMR score of the `i`th document in `bag` relative to `queries(j)`.

`scores = mmrScores( ___ ,lambda)` also specifies the trade off between relevance and redundancy.

Examples

Relevance to Query

Create an array of input documents.

```
str = [
    "the quick brown fox jumped over the lazy dog"
    "the fast fox jumped over the lazy dog"
    "the dog sat there and did nothing"
    "the other animals sat there watching"];
documents = tokenizedDocument(str)
```

```
documents = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    8 tokens: the fast fox jumped over the lazy dog
    7 tokens: the dog sat there and did nothing
    6 tokens: the other animals sat there watching
```

Create an array of query documents.

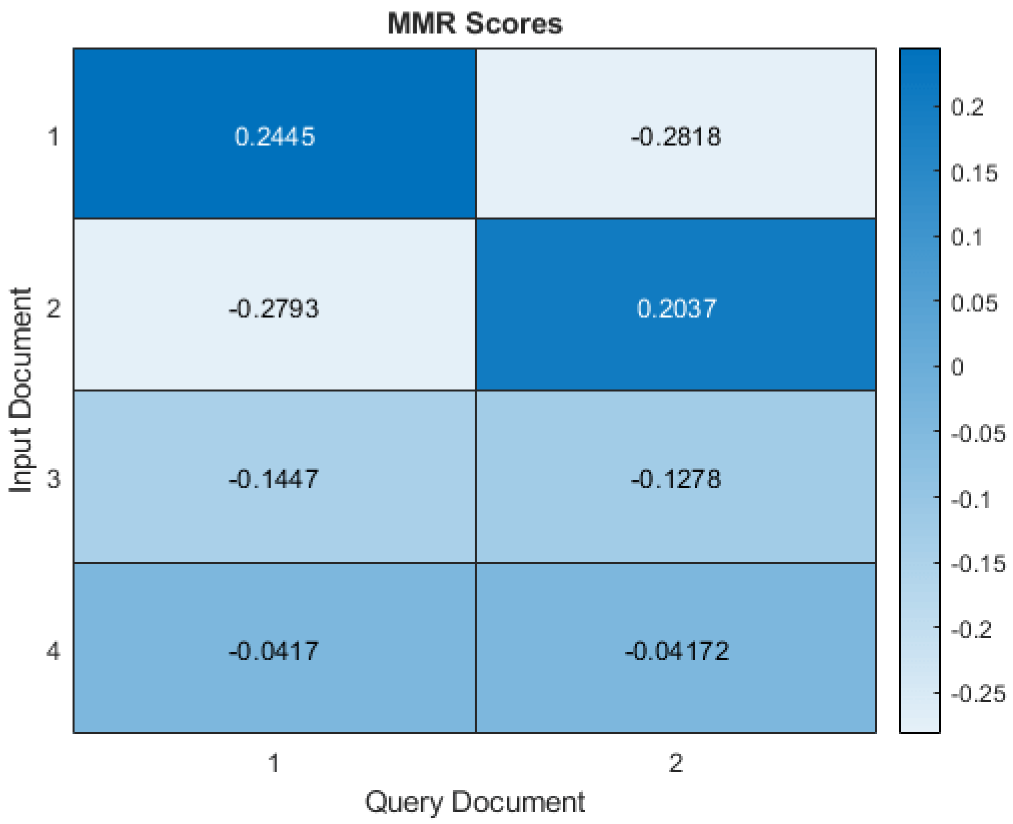
```
str = [
    "a brown fox leaped over the lazy dog"
    "another fox leaped over the dog"];
queries = tokenizedDocument(str)
queries = 
  2x1 tokenizedDocument:

    8 tokens: a brown fox leaped over the lazy dog
    6 tokens: another fox leaped over the dog
```

Calculate MMR scores using the `mmrScores` function. The output is a sparse matrix.

```
scores = mmrScores(documents,queries);
```

Visualize the MMR scores in a heat map.

```
figure
heatmap(scores);
xlabel("Query Document")
ylabel("Input Document")
title("MMR Scores")
```



Higher scores correspond to stonger relavence to the query documents.

Relevance Versus Redundancy

Create an array of input documents.

```
str = [
    "the quick brown fox jumped over the lazy dog"
    "the quick brown fox jumped over the lazy dog"
    "the fast fox jumped over the lazy dog"
    "the dog sat there and did nothing"
    "the other animals sat there watching"
    "the other animals sat there watching"];
documents = tokenizedDocument(str);
```

Create a bag-of-words model from the input documents.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [6x17 double]
      Vocabulary: [1x17 string]
        NumWords: 17
    NumDocuments: 6
```

Create an array of query documents.

```
str = [
    "a brown fox leaped over the lazy dog"
    "another fox leaped over the dog"];
queries = tokenizedDocument(str)
```

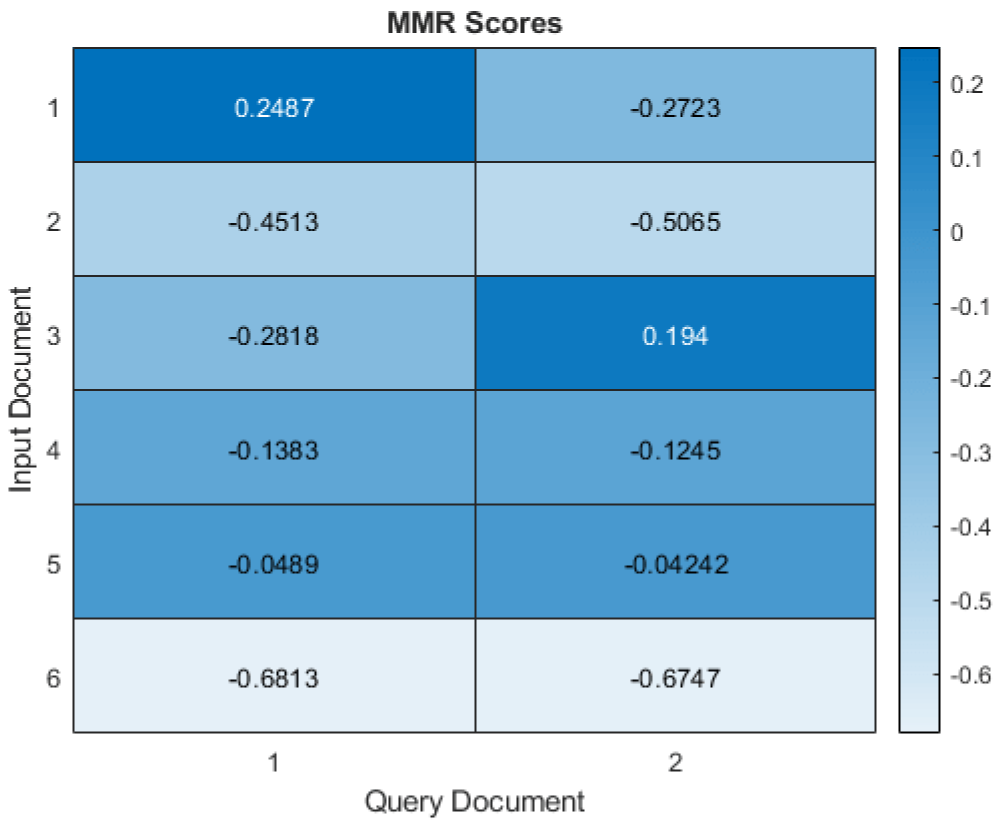
```
queries = 
  2x1 tokenizedDocument:

    8 tokens: a brown fox leaped over the lazy dog
    6 tokens: another fox leaped over the dog
```

Calculate the MMR scores. The output is a sparse matrix.

```
scores = mmrScores(bag,queries);
```

Visualize the MMR scores in a heat map.

```
figure
heatmap(scores);
xlabel("Query Document")
ylabel("Input Document")
title("MMR Scores")
```

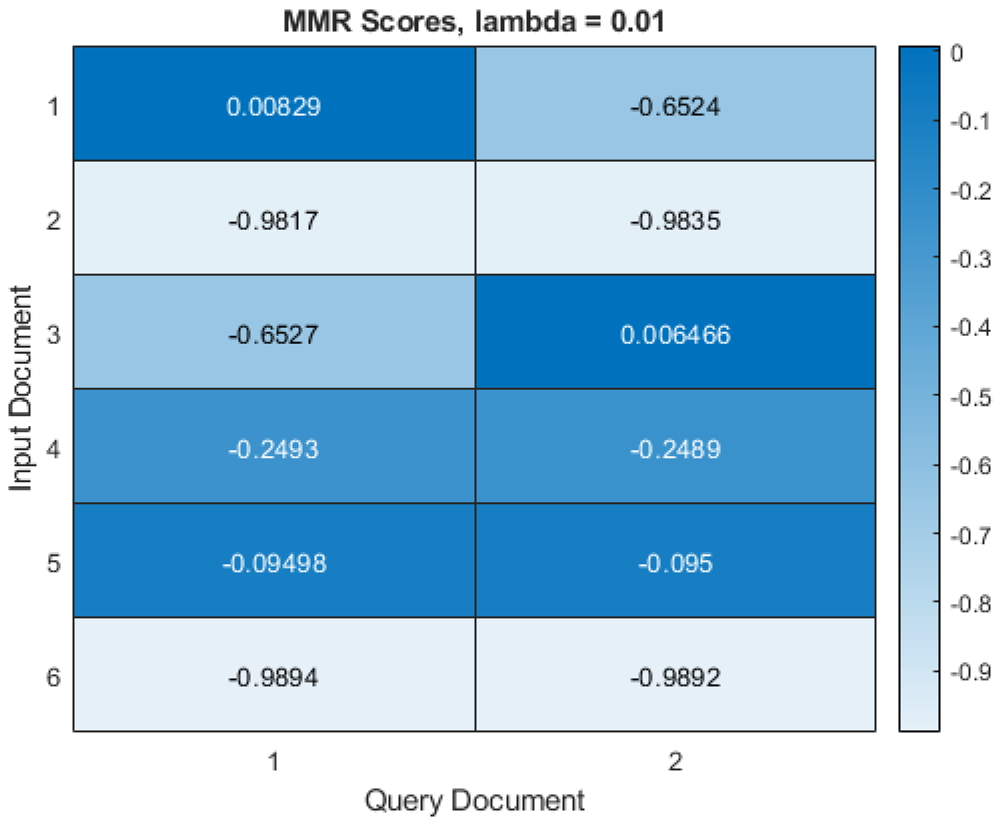


Now calculate the scores again, and set the lambda value to 0.01. When the lambda value is close to 0, redundant documents yield lower scores and diverse (but less query-relevant) documents yield higher scores.

```
lambda = 0.01;
scores = mmrScores(bag,queries,lambda);
```

Visualize the MMR scores in a heat map.

```
figure
heatmap(scores);
xlabel("Query Document")
ylabel("Input Document")
title("MMR Scores, lambda = " + lambda)
```

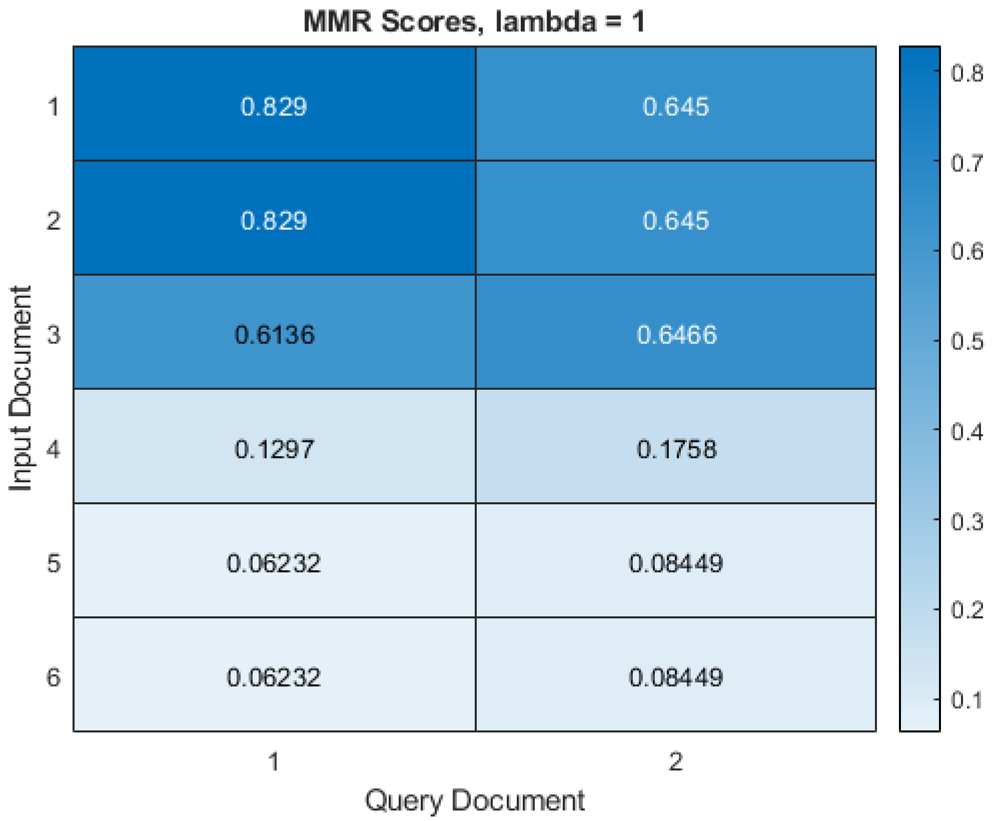


Finally, calculate the scores again and set the lambda value to 1. When the lambda value is 1, the query-relevant documents yield higher scores despite other documents yielding high scores.

```
lambda = 1;
scores = mmrScores(bag,queries,lambda);
```

Visualize the MMR scores in a heat map.

```
figure
heatmap(scores);
xlabel("Query Document")
ylabel("Input Document")
title("MMR Scores, lambda = " + lambda)
```



Input Arguments

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

`bag` — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`queries` — Set of query documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Set of query documents, specified as one of the following:

- A `tokenizedDocument` array
- A 1-by-*N* string array representing a single document, where each element is a word

- A 1-by-*N* cell array of character vectors representing a single document, where each element is a word

To compute term frequency and inverse document frequency statistics, the function encodes `queries` using a bag-of-words model. The model it uses depends on the syntax you call it with. If your syntax specifies the input argument `documents`, then it uses `bagOfWords(documents)`. If your syntax specifies `bag`, then the function encodes `queries` using `bag` then uses the resulting tf-idf matrix.

`lambda` — Trade off between relevance and redundancy
0.3 (default) | nonnegative scalar

Trade off between relevance and redundancy, specified as a nonnegative scalar.

When `lambda` is close to 0, redundant documents yield lower scores and diverse (but less query-relevant) documents yield higher scores. If `lambda` is 1, then query-relevant documents yield higher scores despite other documents yielding high scores.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`scores` — MMR scores
vector

MMR scores, returned as an *N1*-by-*N2* matrix, where `scores(i,j)` is the MMR score of `documents(i)` relative to `j`th query document, and *N1* and *N2* are the number of input and query documents, respectively.

A document has a high MMR score if it is both relevant to the query and has minimal similarity relative to the other documents.

References

[1] Carbonell, Jaime G., and Jade Goldstein. "The use of MMR, diversity-based reranking for reordering documents and producing summaries." In *SIGIR*, vol. 98, pp. 335-336. 1998.

See Also
`bleuEvaluationScore` | `bm25Similarity` | `cosineSimilarity` | `extractSummary` | `lexrankScores` | `rougeEvaluationScore` | `textrankScores` | `tokenizedDocument`

Topics
“Sequence-to-Sequence Translation Using Attention”

Introduced in R2020a

textanalytics.unicode.nfd

Unicode decomposed normalized form (NFD)

Syntax

```
newStr = textanalytics.unicode.nfd(str)
```

Description

`newStr = textanalytics.unicode.nfd(str)` normalizes the string `str` to the Unicode canonical decomposition form (NFD).

Examples

Normalize to Unicode Canonical Decomposition Form

Strings that look identical can have different underlying representations. The Unicode canonical decomposition form (NFD) ensures that equivalent strings have a unique binary representation. This is useful when strings contain accented characters which can have several ways to represent them.

Consider the string `"jalapeño"` which contains 8 letters.

```
str = "jalapeño";
strlength(str)
```

```
ans = 8
```

Normalize the string using the `textanalytics.unicode.nfd` function. Depending on your system, the output string may appear to be identical to the input string.

```
newStr = textanalytics.unicode.nfd(str)
```

```
newStr = 
"jalapeño"
```

View the number of code points in the new string.

```
strlength(newStr)
```

```
ans = 9
```

Notice that the normalized representation includes one extra code point. In this case, the function splits the accented letter `"ñ"` into two separate code points. Extract the 7th and 8th code points in the normalized string. Depending on your system, the output may appear to be a single character.

```
extractBetween(newStr,7,8)
```

```
ans = 
"ñ"
```

Check that the strings `str` and `newStr` are equal using the == operator. The operator returns false because the strings have different underlying representations.

```
tf = str == newStr
```

```
tf = logical
   0
```

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

Example: `["An example of a short sentence."; "A second short sentence."]`

Data Types: `string` | `char` | `cell`

Output Arguments

`newStr` — Output text
string array | character vector | cell array of character vectors

Output text, returned as a string array, a character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

References

[1] *Unicode Standard Annex #15 Unicode Normalization Forms* https://unicode.org/reports/tr15/

See Also

`UTF32` | `characterCategories` | `hex` | `tokenizedDocument`

Topics
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Language Considerations"

Introduced in R2021a

normalizeWords

Stem or lemmatize words

Syntax

```
updatedDocuments = normalizeWords(documents)

updatedWords = normalizeWords(words)
updatedWords = normalizeWords(words,'Language',language)

___ = normalizeWords(___,'Style',style)
```

Description

Use `normalizeWords` to reduce words to a root form. To *lemmatize* English words (reduce them to their dictionary forms), set the `'Style'` option to `'lemma'`.

The function supports English, Japanese, German, and Korean text.

`updatedDocuments = normalizeWords(documents)` reduces the words in `documents` to a root form. For English and German text, the function, by default, stems the words using the Porter stemmer for English and German text respectively. For Japanese and Korean text, the function, by default, lemmatizes the words using the MeCab tokenizer.

`updatedWords = normalizeWords(words)` reduces each word in the string array `words` to a root form.

`updatedWords = normalizeWords(words,'Language',language)` reduces the words and also specifies the word language.

`___ = normalizeWords(___,'Style',style)` also specifies normalization style. For example, `normalizeWords(documents,'Style','lemma')` lemmatizes the words in the input documents.

Examples

Stem Words in Documents

Stem the words in a document array using the Porter stemmer.

```
documents = tokenizedDocument([
    "a strongly worded collection of words"
    "another collection of words"]);
newDocuments = normalizeWords(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: a strongli word collect of word
    4 tokens: anoth collect of word
```

Stem Words in String Array

Stem the words in a string array using the Porter stemmer. Each element of the string array must be a single word.

```
words = ["a" "strongly" "worded" "collection" "of" "words"];
newWords = normalizeWords(words)
```

```
newWords = 1x6 string
    "a"    "strongli"    "word"    "collect"    "of"    "word"
```

Lemmatize Words in Documents

Lemmatize the words in a document array.

```
documents = tokenizedDocument([
    "I am building a house."
    "The building has two floors."]);
newDocuments = normalizeWords(documents,'Style','lemma')
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: i be build a house .
    6 tokens: the build have two floor .
```

To improve the lemmatization, first add part-of-speech details to the documents using the `addPartOfSpeechDetails` function. For example, if the documents contain part-of-speech details, then `normalizeWords` reduces the only verb "building" and not the noun "building".

```
documents = addPartOfSpeechDetails(documents);
newDocuments = normalizeWords(documents,'Style','lemma')
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: i be build a house .
    6 tokens: the building have two floor .
```

Lemmatize Japanese Text

Tokenize Japanese text using the `tokenizedDocument` function. The function automatically detects Japanese text.

```
str = [
    "空に星が輝き、瞬いている。"
    "空の星が輝きを増している。"
    "駅までは遠くて、歩けない。"
    "遠くの駅まで歩けない。"];
documents = tokenizedDocument(str);
```

Lemmatize the tokens using `normalizeWords`.

```
documents = normalizeWords(documents)
```

```
documents = 
  4x1 tokenizedDocument:

    10 tokens: 空 に 星 が 輝く 、 瞬く て いる 。
    10 tokens: 空 の 星 が 輝き を 増す て いる 。
     9 tokens: 駅 まで は 遠い て 、 歩ける ない 。
     7 tokens: 遠く の 駅 まで 歩ける ない 。
```

Stem German Text

Tokenize German text using the `tokenizedDocument` function. The function automatically detects German text.

```
str = [
    "Guten Morgen. Wie geht es dir?"
    "Heute wird ein guter Tag."];
documents = tokenizedDocument(str);
```

Stem the tokens using `normalizeWords`.

```
documents = normalizeWords(documents)
```

```
documents = 
  2x1 tokenizedDocument:

    8 tokens: gut morg . wie geht es dir ?
    6 tokens: heut wird ein gut tag .
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

`style` — Normalization style
`'stem'` | `'lemma'`

Normalization style, specified as one of the following:

- `'stem'` – Stem words using the Porter stemmer. This option supports English and German text only. For English and German text, this value is the default.
- `'lemma'` – Extract the dictionary form of each word. This option supports English, Japanese, and Korean text only. If a word is not in the internal dictionary, then the function outputs the word unchanged. For English text, the output is lowercase. For Japanese and Korean text, this value is the default.

The function only normalizes tokens with type `'letters'` and `'other'`. For more information on token types, see `tokenDetails`.

Tip For English text, to improve lemmatization of words in documents, first add part-of-speech details using the `addPartOfSpeechDetails` function.

`language` — Word language
`'en'` | `'de'`

Word language, specified as one of the following:

- `'en'` – English language
- `'de'` – German language

If you do not specify language, then the software detects the language automatically. To lemmatize Japanese or Korean text, use `tokenizedDocument` input.

Data Types: `char` | `string`

Output Arguments

`updatedDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array.

`updatedWords` — Updated words
string array | character vector | cell array of character vectors

Updated words, returned as a string array, character vector, or cell array of character vectors. `words` and `updatedWords` have the same data type.

Algorithms

Language Details

`tokenizedDocument` objects contain details about the tokens including language details. The language details of the input documents determine the behavior of `normalizeWords`. The `tokenizedDocument` function, by default, automatically detects the language of the input text. To specify the language details manually, use the `'Language'` name-value pair argument of `tokenizedDocument`. To view the token details, use the `tokenDetails` function.

Compatibility Considerations

`normalizeWords` skips complex tokens
Behavior changed in R2018b

Starting in R2018b, for `tokenizedDocument` input, `normalizeWords` normalizes tokens with type `'letters'` or `'other'` only. This behavior prevents the function from affecting complex tokens such as URLs and email-addresses.

In previous versions, `normalizeWords` normalizes all tokens. To reproduce this behavior, use the command `updatedDocuments = docfun(@(str) normalizeWords(str),documents)`.

See Also
`addLemmaDetails` | `addPartOfSpeechDetails` | `bagOfNgrams` | `bagOfWords` | `removeLongWords` | `removeShortWords` | `removeStopWords` | `removeWords` | `stopWords` | `tokenDetails` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Language Considerations"
"Japanese Language Support"
"German Language Support"

Introduced in R2017b

plus, +

Append documents

Syntax

```
newDocuments = documents1 + documents2
newDocuments = plus(documents1,documents2)
```

Description

`newDocuments = documents1 + documents2` appends the documents in `documents2` to the documents in `documents1`.

`newDocuments = plus(documents1,documents2)` is equivalent to `newDocuments = documents1 + documents2`.

Examples

Append Documents

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create arrays containing the first 5 and second 5 sonnets.

```
documents1 = documents(1:5)
```

```
documents1 = 
  5x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youtl
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair tl
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
```

```
documents2 = documents(6:10)
```

```
documents2 = 
  5x1 tokenizedDocument:

    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lovs
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt hap
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art belo
```

Append the second 5 sonnets to the first 5 sonnets.

```
newDocuments = documents1 + documents2
```

```
newDocuments = 
  5x1 tokenizedDocument:

    138 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    135 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy yout
    135 tokens: look thy glass tell face thou viewest time face form another whose fresh repair t
    141 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    130 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
```

Input Arguments

`documents1` — Input documents
array of tokenized documents

Input documents, specified as a `tokenizedDocument` array. `documents1` and `documents2` must be the same size.

`documents2` — Input documents
array of tokenized documents

Input documents, specified as a `tokenizedDocument` array. `documents1` and `documents2` must be the same size.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

See Also

`addPartOfSpeechDetails` | `addSentenceDetails` | `bagOfNgrams` | `bagOfWords` | `docfun` | `eraseURLs` | `normalizeWords` | `replace` | `tokenDetails` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

predict

Predict top LDA topics of documents

Syntax

```
topicIdx = predict(ldaMdl,documents)
topicIdx = predict(ldaMdl,bag)
topicIdx = predict(ldaMdl,counts)
[topicIdx,score] = predict( ___ )
 ___  = predict( ___ ,Name,Value)
```

Description

`topicIdx = predict(ldaMdl,documents)` returns the LDA topic indices with the largest probabilities for `documents` based on the LDA model `ldaMdl`.

`topicIdx = predict(ldaMdl,bag)` returns the LDA topic indices with the largest probabilities for the documents represented by a bag-of-words or bag-of-n-grams model.

`topicIdx = predict(ldaMdl,counts)` returns the LDA topic indices with the largest probabilities for the documents represented by a matrix of word counts.

`[topicIdx,score] = predict( ___ )` also returns a matrix of posterior probabilities `score`.

`___ = predict( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Predict Top LDA Topics of Documents

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:
          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LDA model with 20 topics.

```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0612641 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.26 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.06 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.06 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.06 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.06 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.06 | 6.4180e-05 |  7.291e+02 |         5.000 |             0 |
=====================================================================================

mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Predict the top topics for an array of new documents.

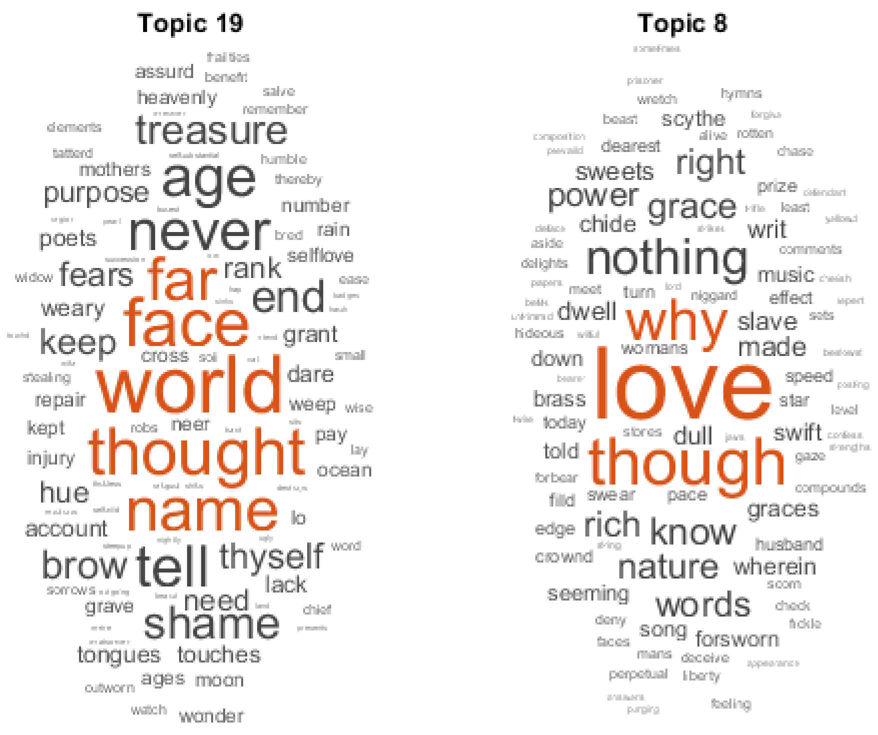
```
newDocuments = tokenizedDocument([
    "what's in a name? a rose by any other name would smell as sweet."
    "if music be the food of love, play on."]);
topicIdx = predict(mdl,newDocuments)
```

```
topicIdx = 2×1

    19
     8
```

Visualize the predicted topics using word clouds.

```
figure
subplot(1,2,1)
wordcloud(mdl,topicIdx(1));
```

```
title("Topic " + topicIdx(1))
subplot(1,2,2)
wordcloud(mdl,topicIdx(2));
title("Topic " + topicIdx(2))
```



Predict Top LDA Topics of Word Count Matrix

Load the example data. `sonnetsCounts.mat` contains a matrix of word counts and a corresponding vocabulary of preprocessed versions of Shakespeare's sonnets.

```
load sonnetsCounts.mat
size(counts)
```

```
ans = 1×2

         154        3092
```

Fit an LDA model with 20 topics. To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
numTopics = 20;
mdl = fitlda(counts,numTopics)
```

```
Initial topic assignments sampled in 0.0587258 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.02 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.06 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.06 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.05 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.07 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.06 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.06 | 6.4180e-05 |  7.291e+02 |         5.000 |             0 |
=====================================================================================
```

```
mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Predict the top topics for the first 5 documents in `counts`.

```
topicIdx = predict(mdl,counts(1:5,:))
```

```
topicIdx = 5×1

     3
    15
    19
     3
    14
```

Calculate Topic Prediction Scores

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LDA model with 20 topics. To suppress verbose output, set `'Verbose'` to 0.

```
numTopics = 20;
mdl = fitlda(bag,numTopics,'Verbose',0);
```

Predict the top topics for a new document. Specify the iteration limit to be 200.

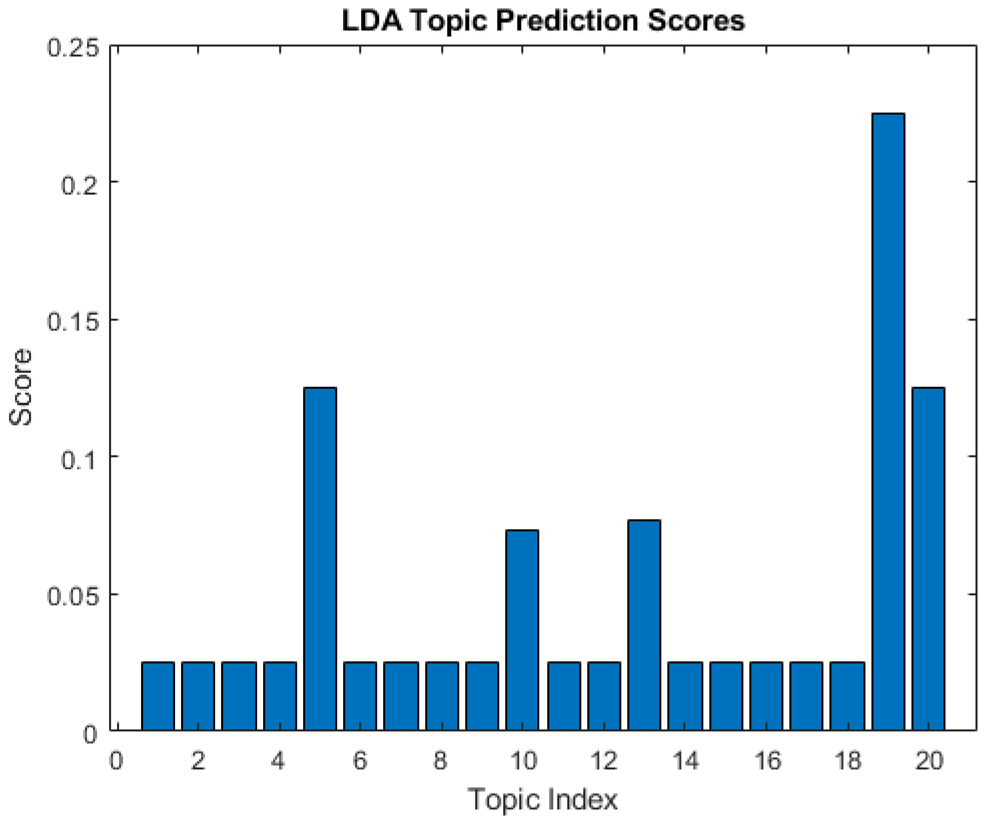
```
newDocument = tokenizedDocument("what's in a name? a rose by any other name would smell as sweet
iterationLimit = 200;
[topicIdx,scores] = predict(mdl,newDocument, ...
    'IterationLimit',iterationLimit)
```

```
topicIdx = 19
```

```
scores = 1×20

    0.0250    0.0250    0.0250    0.0250    0.1250    0.0250    0.0250    0.0250    0.0250    0.0
```

View the prediction scores in a bar chart.

```
figure
bar(scores)
title("LDA Topic Prediction Scores")
xlabel("Topic Index")
ylabel("Score")
```



Input Arguments

`ldaMdl` — Input LDA model
`ldaModel` object

Input LDA model, specified as an `ldaModel` object.

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is a `tokenizedDocument`, then it must be a column vector. If `documents` is a string array or a cell array of character vectors, then it must be a row of the words of a single document.

Tip To ensure that the function does not discard useful information, you must first preprocess the input documents using the same steps used to preprocess the documents used to train the model.

`bag` — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`counts` — Frequency counts of words
matrix of nonnegative integers

Frequency counts of words, specified as a matrix of nonnegative integers. If you specify `'DocumentsIn'` to be `'rows'`, then the value `counts(i,j)` corresponds to the number of times the *j*th word of the vocabulary appears in the *i*th document. Otherwise, the value `counts(i,j)` corresponds to the number of times the *i*th word of the vocabulary appears in the *j*th document.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IterationLimit',200` specifies the iteration limit to be 200.

`DocumentsIn` — Orientation of documents
`'rows'` (default) | `'columns'`

Orientation of documents in the word count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Input is a matrix of word counts with rows corresponding to documents.
- `'columns'` – Input is a transposed matrix of word counts with columns corresponding to documents.

This option only applies if you specify the input documents as a matrix of word counts.

Note If you orient your word count matrix so that documents correspond to columns and specify `'DocumentsIn','columns'`, then you might experience a significant reduction in optimization-execution time.

`IterationLimit` — Maximum number of iterations
`100` (default) | positive integer

Maximum number of iterations, specified as the comma-separated pair consisting of `'IterationLimit'` and a positive integer.

Example: `'IterationLimit',200`

`LogLikelihoodTolerance` — Relative tolerance on log-likelihood
`0.0001` (default) | positive scalar

Relative tolerance on log-likelihood, specified as the comma-separated pair consisting of `'LogLikelihoodTolerance'` and a positive scalar. The optimization terminates when this tolerance is reached.

Example: `'LogLikelihoodTolerance',0.001`

Output Arguments

`topicIdx` — Predicted topic indices
vector of numeric indices

Predicted topic indices, returned as a vector of numeric indices.

`score` — Predicted topic probabilities
matrix

Predicted topic probabilities, returned as a `D`-by-`K` matrix, where `D` is the number of input documents and `K` is the number of topics in the LDA model. `score(i,j)` is the probability that topic `j` appears in document `i`. Each row of `score` sums to one.

See Also

`bagOfWords` | `fitlda` | `ldaModel` | `logp` | `transform` | `wordcloud`

Topics
"Analyze Text Data Using Topic Models"
"Prepare Text Data for Analysis"
"Extract Text Data from Files"

Introduced in R2017b

rakeKeywords

Extract keywords using RAKE

Syntax

```
tbl = rakeKeywords(documents)
tbl = rakeKeywords(documents,Name,Value)
```

Description

`tbl = rakeKeywords(documents)` extracts keywords and respective scores using the Rapid Automatic Keyword Extraction (RAKE) algorithm. The function supports English, Japanese, German, and Korean text. To learn how to use `rakeKeywords` for other languages, see “Language Considerations” on page 1-268.

`tbl = rakeKeywords(documents,Name,Value)` specifies additional options using one or more name-value pair arguments.

Tip The `rakeKeywords` function, by default, extracts keywords using stop words and punctuation characters. When using the default values for the `'Delimiters'` and `'MergingDelimiters'` options, do not remove stop words or punctuation characters from the input text.

Examples

Extract Keywords Using RAKE

Create an array of tokenized documents containing the text data.

```
textData = [
    "MATLAB provides tools for scientists and engineers. MATLAB is used by scientists and enginee
    "Analyze text and images. You can import text and images."
    "Analyze text and images. Analyze text, images, and videos in MATLAB."];
documents = tokenizedDocument(textData);
```

Extract the keywords using the `rakeKeywords` function.

```
tbl = rakeKeywords(documents)
```

```
tbl=12×3 table
                    Keyword                         DocumentNumber    Score
    _________________________________________    ______________    _____

    "MATLAB"         "provides"     "tools"             1            8  
    "MATLAB"         ""             ""                  1            2  
    "scientists"     "and"          "engineers"         1            2  
    "engineers"      ""             ""                  1            1  
    "scientists"     ""             ""                  1            1  
    "Analyze"        "text"         ""                  2            4  
    "import"         "text"         ""                  2            4  
    "images"        ""            ""                    2             1
    "Analyze"       "text"        ""                    3             4
    "MATLAB"        ""            ""                    3             1
    "images"        ""            ""                    3             1
    "videos"        ""            ""                    3             1
```

If a keyword contains multiple words, then the *i*th element of the string array corresponds to the *i*th word of the keyword. If the keyword has fewer words that the longest keyword, then remaining entries of the string array are the empty string `""`.

For readability, transform the multi-word keywords into a single sting using the `join` and `strip` functions.

```
if size(tbl.Keyword,2) > 1
    tbl.Keyword = strip(join(tbl.Keyword));
end
tbl
```

```
tbl=12×3 table
             Keyword               DocumentNumber    Score
    ___________________________    ______________    _____

    "MATLAB provides tools"              1             8
    "MATLAB"                             1             2
    "scientists and engineers"           1             2
    "engineers"                          1             1
    "scientists"                         1             1
    "Analyze text"                       2             4
    "import text"                        2             4
    "images"                             2             1
    "Analyze text"                       3             4
    "MATLAB"                             3             1
    "images"                             3             1
    "videos"                             3             1
```

Specify Maximum Number of Keywords Per Document

Create an array of tokenized document containing the text data.

```
textData = [
    "MATLAB provides tools for scientists and engineers. MATLAB is used by scientists and enginee
    "Analyze text and images. You can import text and images."
    "Analyze text and images. Analyze text, images, and videos in MATLAB."];
documents = tokenizedDocument(textData);
```

Extract the top two keywords using the `rakeKeywords` function and setting the `'MaxNumKeywords'` option to `2`.

```
tbl = rakeKeywords(documents,'MaxNumKeywords',2)
```

```
tbl=6×3 table
                 Keyword                   DocumentNumber    Score
    ___________________________________    ______________    _____
    "MATLAB"     "provides"    "tools"          1             8
    "MATLAB"     ""            ""               1             2
    "Analyze"    "text"        ""               2             4
    "import"     "text"        ""               2             4
    "Analyze"    "text"        ""               3             4
    "MATLAB"     ""            ""               3             1
```

If a keyword contains multiple words, then the *i*th element of the string array corresponds to the *i*th word of the keyword. If the keyword has fewer words that the longest keyword, then remaining entries of the string array are the empty string `""`.

For readability, transform the multi-word keywords into a single sting using the `join` and `strip` functions.

```
if size(tbl.Keyword,2) > 1
    tbl.Keyword = strip(join(tbl.Keyword));
end
tbl
```

```
tbl=6×3 table
            Keyword            DocumentNumber    Score
    _______________________    ______________    _____

    "MATLAB provides tools"          1             8
    "MATLAB"                         1             2
    "Analyze text"                   2             4
    "import text"                    2             4
    "Analyze text"                   3             4
    "MATLAB"                         3             1
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `rakeKeywords(documents,'MaxNumKeywords',20)` returns at most 20 keywords per document.

`MaxNumKeywords` — Maximum number of keywords to return per document
`Inf` (default) | positive integer

Maximum number of keywords to return per document, specified as the comma-separated pair consisting of `'MaxNumKeywords'` and a positive integer or `Inf`.

If `MaxNumKeywords` is `Inf`, then the function returns all identified keywords.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Delimiters` — Tokens for splitting documents into keywords
string array | character vector | cell array of character vectors

Tokens for splitting documents into keywords, specified as the comma-separated pair consisting of `'Delimiters'` and a string array, a character vector, or a cell array of character vectors. If `Delimiters` is a character vector, then it must represent a single delimiter.

The default list of delimiters is a list of punctuation characters.

If multiple candidate keywords appear in a document separated only by merging delimiters, then the function merges those keywords and the merging delimiters into a single keyword.

To specify delimiters for merging, use the `'MergingDelimiters'` option.

Delimiter matching is case insensitive.

Data Types: `char` | `string` | `cell`

`MergingDelimiters` — Delimiters also used for merging keywords
string array | character vector | cell array of character vectors

Delimiters also used for merging keywords, specified as the comma-separated pair consisting of `'MergingDelimiters'` and a string array, a character vector, or a cell array of character vectors. If `MergingDelimiters` is a character vector, then it must represent a single delimiter.

The default list of merging delimiters is the list of stop words given by the `stopWords` function.

If multiple candidate keywords appear in a document separated only by merging delimiters, then the function merges those keywords and the merging delimiters into a single keyword.

To specify delimiters that should not be used for merging, use the `'Delimiters'` option.

Delimiter matching is case insensitive.

Data Types: `char` | `string` | `cell`

Output Arguments

`tbl` — Extracted keywords and scores
table

Extracted keywords and scores, returned as a table with the following variables:

- `Keyword` – Extracted keyword, specified as a 1-by-`maxNgramLength` string array, where `maxNgramLength` is the number of words in the longest keyword.
- `DocumentNumber` – Document number containing the corresponding keyword.
- `Score` – Score of keyword.

If multiple candidate keywords appear in a document separated only by merging delimiters, then the function merges those keywords and the merging delimiters into a single keyword.

If a keyword contains multiple words, then the *i*th element of the corresponding string array corresponds to the *i*th word of the keyword. If the keyword has fewer words that the longest keyword, then remaining entries of the string array are the empty string `""`.

For more information, see “Rapid Automatic Keyword Extraction” on page 1-268.

More About

Language Considerations

The `rakeKeywords` function supports English, Japanese, German, and Korean text only.

The `rakeKeywords` function extracts keywords using a delimiter-based approach to identify candidate keywords. The function, by default, uses punctuation characters and the stop words given by the `stopWords` with language given by the language details of the input documents as delimiters.

For other languages, specify an appropriate set of delimiters using the `'Delimiters'` and `'MergingDelimiters'` options.

Tips

- You can experiment with different keyword extraction algorithms to see what works best with your data. Because the RAKE keywords algorithm uses a delimiter-based approach to extract candidate keywords, the extracted keywords can be very long. Alternatively, you can try extracting keywords using TextRank algorithm which starts with individual tokens as candidate keywords and then merges them when appropriate. To extract keywords using TextRank, use the `textrankKeywords` function. To learn more, see “Extract Keywords from Text Data Using TextRank”.

Algorithms

Rapid Automatic Keyword Extraction

For each document, the `rakeKeywords` function extracts keywords independently using the following steps based on [1]:

1. Determine candidate keywords:

   - Extract sequences of tokens between the delimiters specified by the `'Delimiters'` and `'MergingDelimiters'` options. The function treats each sequence as a single candidate keyword.

2. Calculate scores for the candidate keywords:

   - Create an undirected, unweighted graph with nodes corresponding to the individual tokens in the candidate keywords.
   - Add edges between nodes where tokens co-occur in a candidate keyword, including self co-occurrences, weighted by the number of candidate keywords containing that co-occurrence.

   - Score each token using the formula `deg(token) / freq(token)`, where `deg(token)` is the number of edges for the specified token and `freq(token)` is the number of times that the specified token occurs in the document.
   - For each candidate keyword, assign a score given by the sum of scores of the contained tokens.
3. Extract top keywords from candidates:
   - If there are multiple instances of the same pair of candidate keywords separated by the same single merging delimiter, then merge the candidate keywords and the delimiter into a single keyword and sum the corresponding scores.
   - Return the top *k* keywords, where *k* is given by the `'MaxNumKeywords'` option.

Language Details

`tokenizedDocument` objects contain details about the tokens including language details. The language details of the input documents determine the behavior of `rakeKeywords`. The `tokenizedDocument` function, by default, automatically detects the language of the input text. To specify the language details manually, use the `'Language'` name-value pair argument of `tokenizedDocument`. To view the token details, use the `tokenDetails` function.

References

[1] Rose, Stuart, Dave Engel, Nick Cramer, and Wendy Cowley. "Automatic keyword extraction from individual documents." *Text mining: applications and theory* 1 (2010): 1-20.

See Also

`extractSummary` | `textrankKeywords` | `tokenizedDocument`

Topics
“Extract Keywords from Text Data Using RAKE”
“Extract Keywords from Text Data Using TextRank”

Introduced in R2020b

rangesearch

Find nearest neighbors by edit distance range

Syntax

```
idx = rangesearch(eds,words,maxDist)
[idx,d] = rangesearch(eds,words,maxDist)
```

Description

`idx = rangesearch(eds,words,maxDist)` finds all the words in `eds` that are within distance `maxDist` of the words in `words`.

`[idx,d] = rangesearch(eds,words,maxDist)` also returns the edit distances of the corresponding words.

Examples

Find Nearest Neighbors in Range

Create an edit distance searcher and specify a maximum edit distance of 3.

```
vocabulary = ["MathWorks" "MATLAB" "Simulink" "text" "analytics" "analysis"];
maxDist = 3;
eds = editDistanceSearcher(vocabulary,maxDist);
```

Find the nearest words to `"test"`, `"analytic"`, and `"analyze"` with edit distance less than or equal to 1.

```
words = ["test" "analytic" "analyze"];
maxDist = 1;
idx = rangesearch(eds,words,maxDist)
```

```
idx=3×1 cell array
    {[       4]}
    {[       5]}
    {1x0 double}
```

For `"analyze"`, there are no words in the searcher within the specified range. For `"test"` and `"analytic"`, there is one result each. View the corresponding word for `"test"` using the returned index.

```
nearestWords = eds.Vocabulary(idx{2})
```

```
nearestWords = 
"analytics"
```

Find the nearest words to `"test"`, `"analytic"`, and `"analyze"` with edit distance less than or equal to 3 and their corresponding edit distances.

```
words = ["test" "analytic" "analyze"];
maxDist = 3;
[idx,d] = rangesearch(eds,words,maxDist)
```

```
idx=3×1 cell array
    {[  4]}
    {[5 6]}
    {[  6]}


d=3×1 cell array
    {[  1]}
    {[1 2]}
    {[  3]}
```

For both "test" and "analyze", there is one word in the searcher within the specified range. For "analytic", there are two results. View the corresponding words for "analytic" (the second word) using the returned indices and their edit distances.

```
i = 2;
nearestWords = eds.Vocabulary(idx{i})
```

```
nearestWords = 1x2 string
    "analytics"    "analysis"
```

```
d{i}
```

```
ans = 1×2

     1     2
```

Input Arguments

eds — Edit distance searcher
editDistanceSearcher

Edit distance searcher, specified as an editDistanceSearcher object.

words — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify words as a character vector, then the function treats the argument as a single word.

Data Types: string | char | cell

maxDist — Maximum search distance
non-negative number

Maximum search distance, specified as a non-negative number.

The function finds the indices of the words in eds whose edit distance to the elements of words are fewer than or equal to maxDist, sorted in the ascending order edit distance.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`idx` — Indices of nearest neighbors in searcher
cell array of vectors

Indices of nearest neighbors in the searcher, returned as a cell array of vectors.

`idx{i}` is a vector of indices of the words in `eds` whose edit distance to `words(i)` is less than or equal to `maxDist`, sorted in the ascending order edit distance.

Data Types: `cell`

`d` — Edit distances to neighbors
cell array of vectors

Edit distances to neighbors, returned as a cell array of vectors.

`d{i}` is a vector of edit distances between `words(i)` and the corresponding words in `eds` given by the vocabulary indices `idx{i}`.

Data Types: `cell`

See Also

`correctSpelling` | `editDistance` | `editDistanceSearcher` | `knnsearch` | `splitGraphemes` | `tokenizedDocument`

Topics
"Correct Spelling in Documents"
"Create Extension Dictionary for Spelling Correction"
"Create Custom Spelling Correction Function Using Edit Distance Searchers"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"

Introduced in R2019a

ratioSentimentScores

Sentiment scores with ratio rule

Syntax

```
compoundScores = ratioSentimentScores(documents)
[compoundScores,positiveScores,negativeScores] = ratioSentimentScores(
documents)
___ = ratioSentimentScores( ___ ,Name,Value)
```

Description

Use `ratioSentimentScores` to evaluate sentiment in tokenized text with a ratio rule. The `ratioSentimentScores` function, by default, uses the VADER sentiment lexicon.

`compoundScores = ratioSentimentScores(documents)` returns sentiment scores for tokenized documents based on the ratio of positive and negative tokens. For each document where the ratio of the positive score to negative score is larger than 1, the function returns 1. For each document where the ratio of the negative score to positive score is larger than 1, the function returns -1. Otherwise, the function returns 0.

`[compoundScores,positiveScores,negativeScores] = ratioSentimentScores( documents)` also returns the sums of the positive and negative token scores of the documents respectively.

`___ = ratioSentimentScores( ___ ,Name,Value)` specifies additional options using one or more name-value pairs.

Examples

Evaluate Sentiment in Text

Create a tokenized document.

```
str = [
    "The book was VERY good!!!!"
    "The book was terrible."];
documents = tokenizedDocument(str);
```

Evaluate the sentiment of the tokenized documents. A score of 1 indicates positive sentiment, a score of -1 indicates negative sentiment, and a score of 0 indicates neutral sentiment.

```
compoundScores = ratioSentimentScores(documents)
```

```
compoundScores = 2×1

     1
    -1
```

Evaluate Sentiment Using Custom Lexicon

Sentiment analysis algorithms rely on annotated lists of words called sentiment lexicons. For example, the `ratioSentimentScores` function uses a sentiment lexicon with words annotated with a sentiment score ranging from -1 to 1, where scores close to 1 indicate strong positive sentiment, scores close to -1 indicate strong negative sentiment, and scores close to zero indicate neutral sentiment.

If the sentiment lexicon used by the `ratioSentimentScores` function does not suit the data you are analyzing, for example, if you have a domain-specific data set like medical or engineering data, then you can use your own custom sentiment lexicon. For an example showing how to generate a domain specific sentiment lexicon, see “Generate Domain Specific Sentiment Lexicon”.

Create a tokenized document array containing the text data to analyze.

```
textData = [
    "This company is showing extremely strong growth."
    "This other company is accused of misleading consumers."];
documents = tokenizedDocument(textData);
```

Load the example domain specific lexicon for finance data.

```
filename = "financeSentimentLexicon.csv";
tbl = readtable(filename);
head(tbl)
```

```
ans=8×2 table
         Token          SentimentScore
    _______________    ______________

    {'innovative' }            1
    {'efficiency' }       0.91852
    {'strong'     }       0.82362
    {'efficiently'}       0.81475
    {'creative'   }       0.74264
    {'enhance'    }       0.73791
    {'innovations'}       0.72985
    {'improved'   }       0.71476
```

Evaluate the sentiment using the `ratioSentimentScores` function and specify the custom sentiment lexicon using the `'SentimentLexicon'` option. A score of 1 indicates positive sentiment, a score of -1 indicates negative sentiment, and a score of 0 indicates neutral sentiment.

```
compoundScores = ratioSentimentScores(documents,'SentimentLexicon',tbl)
```

```
compoundScores = 2×1

     1
    -1
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `Threshold,0.5` sets the ratio threshold to 0.5

`SentimentLexicon` — Sentiment lexicon
table

Sentiment lexicon, specified as a table with the following columns:

- `Token` – Token, specified as a string scalar.
- `SentimentScore` – Sentiment score of token, specified as a numeric scalar.

The default sentiment lexicon is the VADER sentiment lexicon.

Data Types: `table`

`Threshold` — Ratio threshold
1 (default) | nonnegative scalar

Ratio threshold, specified as a nonnegative scalar.

If the ratio of the positive score to negative score of `documents(i)` is larger than `Threshold`, then `compoundScores(i)` is 1. If the ratio of the negative score to positive score of `documents(i)` is larger than `Threshold`, then `compoundScores(i)` is -1. Otherwise, `compoundScores(i)` is 0.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`compoundScores` — Compound sentiment scores
numeric vector

Compound sentiment scores, returned as a numeric vector. The function returns one score for each input document.

If the ratio of the positive score to negative score of `documents(i)` is larger than `Threshold`, then `compoundScores(i)` is 1. If the ratio of the negative score to positive score of `documents(i)` is larger than `Threshold`, then `compoundScores(i)` is -1. Otherwise, `compoundScores(i)` is 0.

`positiveScores` — Positive sentiment scores
numeric vector

Positive sentiment scores, returned as a numeric vector. The function returns one score for each input document. The value `positiveScores(i)` corresponds to the positive sentiment score of `documents(i)`.

`negativeScores` — Negative sentiment scores
numeric vector

Negative sentiment scores, returned as a numeric vector. The function returns one score for each input document. The value `negativeScores(i)` corresponds to the negative sentiment score of `documents(i)`.

See Also

`tokenizedDocument` | `vaderSentimentScores`

Topics

"Analyze Sentiment in Text"
"Generate Domain Specific Sentiment Lexicon"
"Train a Sentiment Classifier"
"Create Simple Text Model for Classification"
"Analyze Text Data Containing Emojis"
"Analyze Text Data Using Topic Models"

Introduced in R2019b

readPDFFormData

Read data from PDF forms

Syntax

```
data = readPDFFormData(filename)
data = readPDFFormData(filename,'Password',password)
```

Description

`data = readPDFFormData(filename)` reads the data from a PDF form into a struct.

`data = readPDFFormData(filename,'Password',password)` specifies the password for opening the PDF form.

Examples

Read Data from PDF Form

Read the data from the form fields in `weatherReportForm1.pdf` using `readPDFFormData`. The function returns a struct containing the data from the PDF form fields.

```
filename = "weatherReportForm1.pdf";
data = readPDFFormData(filename)
```

```
data = struct with fields:
         event_type: "Thunderstorm Wind"
    event_narrative: "Large tree down between Plantersville and Nettleton."
```

Read Data From Multiple Forms

Read the data from the form fields in multiple files using a file datastore.

Create a file datastore for the weather reports forms. The forms are named `"weatherReportFormN.pdf"`, where `N` is the number of the form.. Specify the file name using the wildcard "*" to find all file names of this structure. To specify the read function to be `readPDFFormData`, input this function to `fileDatastore` using a function handle.

```
fds = fileDatastore("weatherReportForm*.pdf",'ReadFcn',@readPDFFormData)
```

```
fds = 
  FileDatastore with properties:

                       Files: {
                              ' ...\26\tp5980089c\textanalytics-ex39762425\weatherReportForm1.pd
                              ' ...\26\tp5980089c\textanalytics-ex39762425\weatherReportForm2.pd
                              ' ...\26\tp5980089c\textanalytics-ex39762425\weatherReportForm3.pd
                               ... and 1 more
                              }
```

```
                      Folders: {
                               ' ...\Bdoc21a_1606923_5032\ib8F3FCD\26\tp5980089c\textanalytics-ex
                               }
                  UniformRead: 0
                     ReadMode: 'file'
                    BlockSize: Inf
                   PreviewFcn: @readPDFFormData
      SupportedOutputFormats: [1x16 string]
                      ReadFcn: @readPDFFormData
    AlternateFileSystemRoots: {}
```

Loop over the files in the datastore and read each PDF form.

```
data = [];
while hasdata(fds)
    textData = read(fds);
    data = [data; textData];
end
data
```

```
data=4×1 struct array with fields:
    event_type
    event_narrative
```

Input Arguments

`filename` — Name of file

string scalar | character vector

Name of the file, specified as a string scalar or character vector.

`readPDFFormData` supports AcroForm PDF files (interactive forms) only.

Data Types: `string` | `char`

`password` — Password to open PDF file

string scalar | character vector

Password to open PDF file, specified as a character vector or a string scalar.

Example: `'skroWhtaM'`

Data Types: `string` | `char`

Output Arguments

`data` — Output struct

struct

Output struct. The fields of `data` correspond to the names of the form fields in the PDF. If the form field names are not valid struct field names, then the function automatically edits them to construct valid names.

See Also

`extractFileText` | `extractHTMLText` | `readPDFFormData` | `tokenizedDocument` | `writeTextDocument`

Topics

"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018a

readWordEmbedding

Read word embedding from file

Syntax

```
emb = readWordEmbedding(filename)
```

Description

`emb = readWordEmbedding(filename)` reads the pretrained word embedding stored in text file or zip file `filename`. The input file must be a text file with UTF-8 encoding in either the word2vec or GloVe text embedding format, or a zip file containing a text file of this format.

If the word embedding file contains duplicate words, then the software uses the word vector corresponding to the last duplicate entry.

Examples

Read Word Embedding from Text File

Read the example word embedding. This model was derived by analyzing text from Wikipedia.

```
filename = "exampleWordEmbedding.vec";
emb = readWordEmbedding(filename)
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 50
    Vocabulary: [1x9999 string]
```

Explore the word embedding using `word2vec` and `vec2word`.

```
king = word2vec(emb,"king");
man = word2vec(emb,"man");
woman = word2vec(emb,"woman");
word = vec2word(emb,king - man + woman)
```

```
word = 
"queen"
```

Input Arguments

`filename` — Name of file
string scalar | character vector

Name of the file, specified as a string scalar or character vector.

Data Types: `string` | `char`

Output Arguments

`emb` — Output word embedding
word embedding

Output word embedding, returned as a `wordEmbedding` object.

See Also

`doc2sequence` | `fastTextWordEmbedding` | `tokenizedDocument` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding` | `writeWordEmbedding`

Topics
"Train a Sentiment Classifier"
"Classify Text Data Using Deep Learning"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2017b

regexprep

Replace text in words of documents using regular expression

Syntax

`newDocuments = regexprep(documents,expression,replace)`

Description

Text Analytics Toolbox provides functions for common text preprocessing steps. For example, to remove punctuation and symbol characters, use `erasePunctuation` or to remove stem words using the Porter stemmer, use `normalizeWords`. For more information, see “Text Data Preparation”.

`newDocuments = regexprep(documents,expression,replace)` replaces all occurrences of the regular expression `expression` in the words of `documents` with the text in `replace`.

The function matches each word independently. The match does not have to span the whole word.

Examples

Update Text in Words

Replace words that begin with `"s"`, end `"e"`, and have at least one character between them. To match whole words, use `"^"` to match the start of a word and `"$"` to match the end of the word.

```
documents = tokenizedDocument([ ...
    "an example of a short sentence"
    "a second short sentence"])
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

```
expression = "^s(\w+)e$";
replace = "thing";
newDocuments = regexprep(documents,expression,replace)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short thing
    4 tokens: a second short thing
```

If you do not use `"^"` and `"$"`, then you can match substrings of the words. Replace all vowels with `"_"`.

```
expression = "[aeiou]";
replace = "\_";
newDocuments = regexprep(documents,expression,replace)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: _n _x_mpl_ _f _ sh_rt s_nt_nc_
    4 tokens: _ s_c_nd sh_rt s_nt_nc_
```

Include Captured Tokens in Word Replacement

Replace variations of the word "walk" by capturing the letters that follow "walk".

```
documents = tokenizedDocument([
    "I walk"
    "they walked"
    "we are walking"])
```

```
documents = 
  3x1 tokenizedDocument:

    2 tokens: I walk
    2 tokens: they walked
    3 tokens: we are walking
```

```
expression = "walk(\w*)";
replace = "ascend$1";
newDocuments = regexprep(documents,expression,replace)
```

```
newDocuments = 
  3x1 tokenizedDocument:

    2 tokens: I ascend
    2 tokens: they ascended
    3 tokens: we are ascending
```

Input Arguments

`documents` — Input documents

`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`expression` — Regular expression

character vector | cell array of character vectors | string array

Regular expression, specified as a character vector, a cell array of character vectors, or a string array. Each expression can contain characters, metacharacters, operators, tokens, and flags that specify patterns to match in `str`.

The following tables describe the elements of regular expressions.

Metacharacters

Metacharacters represent letters, letter ranges, digits, and space characters. Use them to construct a generalized pattern of characters.

Metacharacter	Description	Example
`.`	Any single character, including white space	`'..ain'` matches sequences of five consecutive characters that end with `'ain'`.
`[`$c_1c_2c_3$`]`	Any character contained within the square brackets. The following characters are treated literally: `$ | . * + ?` and `-` when not used to indicate a range.	`'[rp.]ain'` matches `'rain'` or `'pain'` or `'.ain'`.
`[^`$c_1c_2c_3$`]`	Any character not contained within the square brackets. The following characters are treated literally: `$ | . * + ?` and `-` when not used to indicate a range.	`'[^*rp]ain'` matches all four-letter sequences that end in `'ain'`, except `'rain'` and `'pain'` and `'*ain'`. For example, it matches `'gain'`, `'lain'`, or `'vain'`.
`[`c_1`-`c_2`]`	Any character in the range of c_1 through c_2	`'[A-G]'` matches a single character in the range of `A` through `G`.
`\w`	Any alphabetic, numeric, or underscore character. For English character sets, `\w` is equivalent to `[a-zA-Z_0-9]`	`'\w*'` identifies a word comprised of any grouping of alphabetic, numeric, or underscore characters.
`\W`	Any character that is not alphabetic, numeric, or underscore. For English character sets, `\W` is equivalent to `[^a-zA-Z_0-9]`	`'\W*'` identifies a term that is not a word comprised of any grouping of alphabetic, numeric, or underscore characters.
`\s`	Any white-space character; equivalent to `[ \f\n\r\t\v]`	`'\w*n\s'` matches words that end with the letter `n`, followed by a white-space character.
`\S`	Any non-white-space character; equivalent to `[^ \f\n\r\t\v]`	`'\d\S'` matches a numeric digit followed by any non-white-space character.
`\d`	Any numeric digit; equivalent to `[0-9]`	`'\d*'` matches any number of consecutive digits.
`\D`	Any nondigit character; equivalent to `[^0-9]`	`'\w*\D\>'` matches words that do not end with a numeric digit.
`\oN` or `\o{N}`	Character of octal value `N`	`'\o{40}'` matches the space character, defined by octal `40`.
`\xN` or `\x{N}`	Character of hexadecimal value `N`	`'\x2C'` matches the comma character, defined by hex `2C`.

Character Representation

Operator	Description
`\a`	Alarm (beep)
`\b`	Backspace

Operator	Description
`\f`	Form feed
`\n`	New line
`\r`	Carriage return
`\t`	Horizontal tab
`\v`	Vertical tab
`\`*`char`*	Any character with special meaning in regular expressions that you want to match literally (for example, use `\\` to match a single backslash)

Quantifiers

Quantifiers specify the number of times a pattern must occur in the matching text.

Quantifier	Number of Times Expression Occurs	Example
`expr*`	0 or more times consecutively.	`'\w*'` matches a word of any length.
`expr?`	0 times or 1 time.	`'\w*(\.m)?'` matches words that optionally end with the extension `.m`.
`expr+`	1 or more times consecutively.	`'<img src="\w+\.gif">'` matches an `<img>` HTML tag when the file name contains one or more characters.
`expr{m,n}`	At least `m` times, but no more than `n` times consecutively. `{0,1}` is equivalent to `?`.	`'\S{4,8}'` matches between four and eight non-white-space characters.
`expr{m,}`	At least `m` times consecutively. `{0,}` and `{1,}` are equivalent to `*` and `+`, respectively.	`'<a href="\w{1,}\.html">'` matches an `<a>` HTML tag when the file name contains one or more characters.
`expr{n}`	Exactly `n` times consecutively. Equivalent to `{n,n}`.	`'\d{4}'` matches four consecutive digits.

Quantifiers can appear in three modes, described in the following table. *q* represents any of the quantifiers in the previous table.

Mode	Description	Example
`expr`*`q`*	Greedy expression: match as many characters as possible.	Given the text `'<tr><td><p>text</p></td>'`, the expression `'</?t.*>'` matches all characters between `<tr` and `/td>`: `'<tr><td><p>text</p></td>'`
`expr`*`q`*`?`	Lazy expression: match as few characters as necessary.	Given the text`'<tr><td><p>text</p></td>'`, the expression `'</?t.*?>'` ends each match at the first occurrence of the closing angle bracket (`>`): `'<tr>'   '<td>'   '</td>'`

Mode	Description	Example
`expr`*q*`+`	Possessive expression: match as much as possible, but do not rescan any portions of the text.	Given the text`'<tr><td><p>text</p></td>'`, the expression `'</?t.*+>'` does not return any matches, because the closing angle bracket is captured using `.*`, and is not rescanned.

Grouping Operators

Grouping operators allow you to capture tokens, apply one operator to multiple elements, or disable backtracking in a specific group.

Grouping Operator	Description	Example
`(expr)`	Group elements of the expression and capture tokens.	`'Joh?n\s(\w*)'` captures a token that contains the last name of any person with the first name `John` or `Jon`.
`(?:expr)`	Group, but do not capture tokens.	`'(?:[aeiou][^aeiou]){2}'` matches two consecutive patterns of a vowel followed by a nonvowel, such as `'anon'`. Without grouping, `'[aeiou][^aeiou]{2}'`matches a vowel followed by two nonvowels.
`(?>expr)`	Group atomically. Do not backtrack within the group to complete the match, and do not capture tokens.	`'A(?>.*)Z'` does not match `'AtoZ'`, although `'A(?:.*)Z'` does. Using the atomic group, `Z` is captured using `.*` and is not rescanned.
`(expr1|expr2)`	Match expression `expr1` or expression `expr2`. If there is a match with `expr1`, then `expr2` is ignored. You can include `?:` or `?>` after the opening parenthesis to suppress tokens or group atomically.	`'(let|tel)\w+'` matches words that start with `let` or `tel`.

Anchors

Anchors in the expression match the beginning or end of the input text or word.

Anchor	Matches the...	Example
`^expr`	Beginning of the input text.	`'^M\w*'` matches a word starting with `M` at the beginning of the text.
`expr$`	End of the input text.	`'\w*m$'` matches words ending with `m` at the end of the text.
`\<expr`	Beginning of a word.	`'\<n\w*'` matches any words starting with `n`.

Anchor	Matches the...	Example
`expr\>`	End of a word.	`'\w*e\>'` matches any words ending with `e`.

Lookaround Assertions

Lookaround assertions look for patterns that immediately precede or follow the intended match, but are not part of the match.

The pointer remains at the current location, and characters that correspond to the `test` expression are not captured or discarded. Therefore, lookahead assertions can match overlapping character groups.

Lookaround Assertion	Description	Example
`expr(?=test)`	Look ahead for characters that match `test`.	`'\w*(?=ing)'` matches terms that are followed by `ing`, such as `'Fly'` and `'fall'` in the input text `'Flying, not falling.'`
`expr(?!test)`	Look ahead for characters that do not match `test`.	`'i(?!ng)'` matches instances of the letter `i` that are not followed by `ng`.
`(?<=test)expr`	Look behind for characters that match `test`.	`'(?<=re)\w*'` matches terms that follow `'re'`, such as `'new'`, `'use'`, and `'cycle'` in the input text `'renew, reuse, recycle'`
`(?<!test)expr`	Look behind for characters that do not match `test`.	`'(?<!\d)(\d)(?!\d)'` matches single-digit numbers (digits that do not precede or follow other digits).

If you specify a lookahead assertion *before* an expression, the operation is equivalent to a logical `AND`.

Operation	Description	Example
`(?=test)expr`	Match both `test` and `expr`.	`'(?=[a-z])[^aeiou]'` matches consonants.
`(?!test)expr`	Match `expr` and do not match `test`.	`'(?![aeiou])[a-z]'` matches consonants.

Logical and Conditional Operators

Logical and conditional operators allow you to test the state of a given condition, and then use the outcome to determine which pattern, if any, to match next. These operators support logical `OR`, and `if` or `if/else` conditions.

Conditions can be tokens, lookaround operators, or dynamic expressions of the form `(?@cmd)`. Dynamic expressions must return a logical or numeric value.

Conditional Operator	Description	Example
`expr1|expr2`	Match expression `expr1` or expression `expr2`. If there is a match with `expr1`, then `expr2` is ignored.	`'(let|tel)\w+'` matches words that start with `let` or `tel`.

Conditional Operator	Description	Example
`(?(cond)expr)`	If condition `cond` is `true`, then match `expr`.	`'(?(?@ispc)[A-Z]:\\)'` matches a drive name, such as `C:\`, when run on a Windows® system.
`(?(cond)expr1|expr2)`	If condition `cond` is `true`, then match `expr1`. Otherwise, match `expr2`.	`'Mr(s?)\..*?(?(1)her|his) \w*'` matches text that includes `her` when the text begins with `Mrs`, or that includes `his` when the text begins with `Mr`.

Token Operators

Tokens are portions of the matched text that you define by enclosing part of the regular expression in parentheses. You can refer to a token by its sequence in the text (an ordinal token), or assign names to tokens for easier code maintenance and readable output.

Ordinal Token Operator	Description	Example
`(expr)`	Capture in a token the characters that match the enclosed expression.	`'Joh?n\s(\w*)'` captures a token that contains the last name of any person with the first name `John` or `Jon`.
`\N`	Match the `N`th token.	`'<(\w+).*>.*</\1>'` captures tokens for HTML tags, such as `'title'` from the text `'<title>Some text</title>'`.
`(?(N)expr1|expr2)`	If the `N`th token is found, then match `expr1`. Otherwise, match `expr2`.	`'Mr(s?)\..*?(?(1)her|his) \w*'` matches text that includes `her` when the text begins with `Mrs`, or that includes `his` when the text begins with `Mr`.

Named Token Operator	Description	Example
`(?<name>expr)`	Capture in a named token the characters that match the enclosed expression.	`'(?<month>\d+)-(?<day>\d+)-(?<yr>\d+)'` creates named tokens for the month, day, and year in an input date of the form `mm-dd-yy`.
`\k<name>`	Match the token referred to by `name`.	`'<(?<tag>\w+).*>.*</\k<tag>>'` captures tokens for HTML tags, such as `'title'` from the text `'<title>Some text</title>'`.
`(?(name)expr1|expr2)`	If the named token is found, then match `expr1`. Otherwise, match `expr2`.	`'Mr(?<sex>s?)\..*?(?(sex)her|his) \w*'` matches text that includes `her` when the text begins with `Mrs`, or that includes `his` when the text begins with `Mr`.

Note If an expression has nested parentheses, MATLAB® captures tokens that correspond to the outermost set of parentheses. For example, given the search pattern `'(and(y|rew))'`, MATLAB creates a token for `'andrew'` but not for `'y'` or `'rew'`.

Dynamic Regular Expressions

Dynamic expressions allow you to execute a MATLAB command or a regular expression to determine the text to match.

The parentheses that enclose dynamic expressions do *not* create a capturing group.

Operator	Description	Example
`(??expr)`	Parse `expr` and include the resulting term in the match expression. When parsed, `expr` must correspond to a complete, valid regular expression. Dynamic expressions that use the backslash escape character (\\) require two backslashes: one for the initial parsing of `expr`, and one for the complete match.	`'^(\d+)((??\\w{$1}))'` determines how many characters to match by reading a digit at the beginning of the match. The dynamic expression is enclosed in a second set of parentheses so that the resulting match is captured in a token. For instance, matching `'5XXXXX'` captures tokens for `'5'` and `'XXXXX'`.
`(??@cmd)`	Execute the MATLAB command represented by `cmd`, and include the output returned by the command in the match expression.	`'(.{2,}).?(??@fliplr($1))'` finds palindromes that are at least four characters long, such as `'abba'`.
`(?@cmd)`	Execute the MATLAB command represented by `cmd`, but discard any output the command returns. (Helpful for diagnosing regular expressions.)	`'\w*?(\w)(?@disp($1))\1\w*'` matches words that include double letters (such as `pp`), and displays intermediate results.

Within dynamic expressions, use the following operators to define replacement text.

Replacement Operator	Description
`$&` or `$0`	Portion of the input text that is currently a match
`` $` ``	Portion of the input text that precedes the current match
`$'`	Portion of the input text that follows the current match (use `$''` to represent `$'`)
`$N`	`N`th token
`$<name>`	Named token
`${cmd}`	Output returned when MATLAB executes the command, `cmd`

Comments

Characters	Description	Example
`(?#comment)`	Insert a comment in the regular expression. The comment text is ignored when matching the input.	`'(?# Initial digit)\<\d\w+'` includes a comment, and matches words that begin with a number.

Search Flags

Search flags modify the behavior for matching expressions. An alternative to using a search flag within an expression is to pass an `option` input argument.

Flag	Description
`(?-i)`	Match letter case (default for `regexp` and `regexprep`).
`(?i)`	Do not match letter case (default for `regexpi`).
`(?s)`	Match dot (`.`) in the pattern with any character (default).
`(?-s)`	Match dot in the pattern with any character that is not a newline character.
`(?-m)`	Match the `^` and `$` metacharacters at the beginning and end of text (default).
`(?m)`	Match the `^` and `$` metacharacters at the beginning and end of a line.
`(?-x)`	Include space characters and comments when matching (default).
`(?x)`	Ignore space characters and comments when matching. Use `'\ '` and `'\#'` to match space and `#` characters.

The expression that the flag modifies can appear either after the parentheses, such as

```
(?i)\w*
```

or inside the parentheses and separated from the flag with a colon (`:`), such as

```
(?i:\w*)
```

The latter syntax allows you to change the behavior for part of a larger expression.

Data Types: `char` | `cell` | `string`

`replace` — Replacement text
character vector | cell array of character vectors | string array

Replacement text, specified as a character vector, a cell array of character vectors, or a string array, as follows:

- If `replace` is a single character vector and `expression` is a cell array of character vectors, then `regexprep` uses the same replacement text for each expression.
- If `replace` is a cell array of `N` character vectors and `expression` is a single character vector, then `regexprep` attempts `N` matches and replacements.
- If both `replace` and `expression` are cell arrays of character vectors, then they must contain the same number of elements. `regexprep` pairs each `replace` element with its corresponding element in `expression`.

The replacement text can include regular characters, special characters (such as tabs or new lines), or replacement operators, as shown in the following tables.

Replacement Operator	Description
`$&` or `$0`	Portion of the input text that is currently a match
`` $` ``	Portion of the input text that precedes the current match
`$'`	Portion of the input text that follows the current match (use `$''` to represent `$'`)
`$N`	Nth token
`$<name>`	Named token
`${cmd}`	Output returned when MATLAB executes the command, `cmd`

Operator	Description
`\a`	Alarm (beep)
`\b`	Backspace
`\f`	Form feed
`\n`	New line
`\r`	Carriage return
`\t`	Horizontal tab
`\v`	Vertical tab
`\`*`char`*	Any character with special meaning in regular expressions that you want to match literally (for example, use `\\` to match a single backslash)

Data Types: `char` | `cell` | `string`

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

Tips

- Text Analytics Toolbox provides functions for common text preprocessing steps. For example, to remove punctuation and symbol characters, use `erasePunctuation` or to remove stem words using the Porter stemmer, use `normalizeWords`. For more information, see “Text Data Preparation”.

See Also

`decodeHTMLEntities` | `erasePunctuation` | `eraseTags` | `eraseURLs` | `normalizeWords` | `removeLongWords` | `removeShortWords` | `removeWords` | `replace` | `tokenizedDocument`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

removeDocument

Remove documents from bag-of-words or bag-of-n-grams model

Syntax

```
newBag = removeDocument(bag,idx)
```

Description

`newBag = removeDocument(bag,idx)` removes the documents with indices specified by `idx` from the bag-of-words or bag-of-n-grams model `bag`. If the removed documents contain words or n-grams that do not appear in the remaining documents, then the function also removes these words or n-grams from `bag`.

Examples

Remove Documents from Bag-of-Words Model

Remove selected documents from a bag-of-words model.

```
documents = tokenizedDocument([ ...
    "an example of a short sentence"
    "a second short sentence"
    "a third example"
    "a final sentence"]);
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [4x9 double]
      Vocabulary: [1x9 string]
        NumWords: 9
    NumDocuments: 4
```

Remove the first and third documents from `bag`.

```
idx = [1 3];
newBag = removeDocument(bag,idx)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [2x5 double]
      Vocabulary: ["a"    "short"    "sentence"    "second"    "final"]
        NumWords: 5
    NumDocuments: 2
```

Remove the same documents using logical indices.

```
idx = logical([1 0 1 0]);
newBag = removeDocument(bag,idx)

newBag = 
  bagOfWords with properties:

          Counts: [2x5 double]
      Vocabulary: ["a"    "short"    "sentence"    "second"    "final"]
        NumWords: 5
    NumDocuments: 2
```

Input Arguments

bag — Input bag-of-words or bag-of-n-grams model
bagOfWords object | bagOfNgrams object

Input bag-of-words or bag-of-n-grams model, specified as a bagOfWords object or a bagOfNgrams object.

idx — Indices of documents to remove
vector of numeric indices | vector of logical indices

Indices of documents to remove, specified as a vector of numeric indices or a vector of logical indices.

Example: [2 4 6]

Example: [0 1 0 1 0 1]

Output Arguments

newBag — Output model
bagOfWords object | bagOfNgrams object

Output model, returned as a bagOfWords object or a bagOfNgrams object. The type of newBag is the same as the type of bag.

See Also
addDocument | bagOfNgrams | bagOfWords | removeEmptyDocuments | tokenizedDocument

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Analyze Text Data Using Topic Models”
“Analyze Text Data Using Multiword Phrases”
“Visualize Text Data Using Word Clouds”
“Classify Text Data Using Deep Learning”

Introduced in R2017b

removeEmptyDocuments

Remove empty documents from tokenized document array, bag-of-words model, or bag-of-n-grams model

Syntax

```
newDocuments = removeEmptyDocuments(documents)
newBag = removeEmptyDocuments(bag)
[ ___ ,idx] = removeEmptyDocuments( ___ )
```

Description

`newDocuments = removeEmptyDocuments(documents)` removes documents which have no words from `documents`.

`newBag = removeEmptyDocuments(bag)` removes documents which have no words or n-grams from the bag-of-words or bag-of-n-grams model `bag`.

`[ ___ ,idx] = removeEmptyDocuments( ___ )` also returns the indices of the removed documents.

Examples

Remove Empty Documents from Array

Remove documents containing no words from an array of tokenized documents.

Create an array of tokenized documents which includes empty documents.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    ""
    "a second short sentence"
    ""])
```

```
documents = 
  4x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    0 tokens:
    4 tokens: a second short sentence
    0 tokens:
```

Remove the empty documents.

```
newDocuments = removeEmptyDocuments(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:
    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

Remove Empty Documents from Bag-of-Words Model

Remove documents containing no words from bag-of-words model.

Create a bag-of-words model from an array of tokenized documents.

```
documents = tokenizedDocument([
    "An example of a short sentence."
    ""
    "A second short sentence."
    ""]);
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [4x9 double]
      Vocabulary: [1x9 string]
        NumWords: 9
    NumDocuments: 4
```

Remove the empty documents from the bag-of-words model.

```
newBag = removeEmptyDocuments(bag)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [2x9 double]
      Vocabulary: [1x9 string]
        NumWords: 9
    NumDocuments: 2
```

Remove Documents and Corresponding Labels

Remove documents containing no words from an array and use the indices of removed documents to remove the corresponding labels also.

Create an array of tokenized documents which includes empty documents.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    ""
    "a second short sentence"
    ""])
```

```
documents = 
  4x1 tokenizedDocument:
    6 tokens: an example of a short sentence
    0 tokens:
    4 tokens: a second short sentence
    0 tokens:
```

Create a vector of labels.

```
labels = ["T"; "F"; "F"; "T"]
```

```
labels = 4x1 string
    "T"
    "F"
    "F"
    "T"
```

Remove the empty documents and get the indices of the removed documents.

```
[newDocuments, idx] = removeEmptyDocuments(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence


idx = 2×1

     2
     4
```

Remove the corresponding labels from `labels`.

```
labels(idx) = []
```

```
labels = 2x1 string
    "T"
    "F"
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`bag` — Input bag-of-words or bag-of-n-grams model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

`newBag` — Output model
`bagOfWords` object | `bagOfNgrams` object

Output model, returned as a `bagOfWords` object or a `bagOfNgrams` object. The type of `newBag` is the same as the type of `bag`.

`idx` — Indices of removed documents
vector of positive integers

Indices of removed documents, returned as a vector of positive integers.

See Also

`addDocument` | `bagOfNgrams` | `bagOfWords` | `removeDocument` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2017b

removeInfrequentNgrams

Remove infrequently seen n-grams from bag-of-n-grams model

Syntax

```
newBag = removeInfrequentNgrams(bag,count)
newBag = removeInfrequentNgrams(bag,count,'NgramLengths',lengths)
newBag = removeInfrequentNgrams( ___ ,'IgnoreCase',true)
```

Description

`newBag = removeInfrequentNgrams(bag,count)` removes the n-grams that appear at most `count` times in total from the bag-of-n-grams model `bag`. The function, by default, is case sensitive.

`newBag = removeInfrequentNgrams(bag,count,'NgramLengths',lengths)` only removes n-grams with lengths specified by `lengths`. The function, by default, is case sensitive.

`newBag = removeInfrequentNgrams( ___ ,'IgnoreCase',true)` removes the n-grams that appear at most `count` times ignoring case. If n-grams differ only by case, then the corresponding counts are merged.

Examples

Remove Infrequent N-Grams from Bag-of-N-Grams Model

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-n-grams model. Specify to count bigrams (pairs of words) and trigrams (triples of words).

```
bag = bagOfNgrams(documents,'NgramLengths',[2 3])
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154x18022 double]
      Vocabulary: [1x3092 string]
          Ngrams: [18022x3 string]
    NgramLengths: [2 3]
       NumNgrams: 18022
    NumDocuments: 154
```

Remove n-grams of any length that appear two or fewer times in total.

```
bag = removeInfrequentNgrams(bag,2)
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154x103 double]
      Vocabulary: [1x73 string]
          Ngrams: [103x3 string]
    NgramLengths: [2 3]
       NumNgrams: 103
    NumDocuments: 154
```

Remove bigrams that appear four or fewer times in total.

```
bag = removeInfrequentNgrams(bag,4,'NgramLengths',2)
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154x41 double]
      Vocabulary: [1x30 string]
          Ngrams: [41x3 string]
    NgramLengths: [2 3]
       NumNgrams: 41
    NumDocuments: 154
```

Input Arguments

bag — Input bag-of-n-grams model
`bagOfNgrams` object

Input bag-of-n-grams model, specified as a `bagOfNgrams` object.

`count` — Count threshold
positive integer

Count threshold, specified as a positive integer. The function removes the n-grams that appear `count` times in total or fewer.

`lengths` — N-gram lengths
positive integer | vector of positive integers

N-gram lengths, specified as a positive integer or a vector of positive integers.

If you specify `lengths`, the function removes infrequent n-grams of the specified lengths only. If you do not specify `lengths`, then the function removes infrequent n-grams regardless of length.

Example: `[1 2 3]`

Output Arguments

newBag — Output bag-of-n-grams model
`bagOfNgrams` object

Output bag-of-n-grams model, returned as a `bagOfNgrams` object.

See Also

`bagOfNgrams` | `bagOfWords` | `removeEmptyDocuments` | `removeInfrequentWords` | `removeNgrams` | `tfidf` | `tokenizedDocument` | `topkngrams`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2018a

removeInfrequentWords

Remove words with low counts from bag-of-words model

Syntax

```
newBag = removeInfrequentWords(bag,count)
newBag = removeInfrequentWords(bag,count,'IgnoreCase',true)
```

Description

`newBag = removeInfrequentWords(bag,count)` removes the words that appear at most `count` times in total from the bag-of-words model `bag`. The function, by default, is case sensitive.

`newBag = removeInfrequentWords(bag,count,'IgnoreCase',true)` removes the words that appear at most `count` times in total ignoring case. If words differ only by case, then the corresponding counts are merged.

Examples

Remove Infrequent Words

Remove the words that appear two times or fewer from a bag-of-words model.

Create a bag-of-words model from an array of tokenized documents.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"
    "another example"
    "a short example"]);
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [4x8 double]
      Vocabulary: [1x8 string]
        NumWords: 8
    NumDocuments: 4
```

Remove the words that appear two times or fewer from the bag-of-words model.

```
count = 2;
newBag = removeInfrequentWords(bag,count)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [4x3 double]
      Vocabulary: ["example"    "a"    "short"]
```

```
       NumWords: 3
   NumDocuments: 4
```

Input Arguments

bag — Input bag-of-words model
`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` object.

`count` — Count threshold to remove words
positive integer

Count threshold to remove words, specified as a positive integer. The function removes the words that appear `count` times in total or fewer.

See Also

`bagOfNgrams` | `bagOfWords` | `removeEmptyDocuments` | `removeInfrequentNgrams` | `removeWords` | `tfidf` | `tokenizedDocument` | `topkwords`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"

Introduced in R2017b

removeLongWords

Remove long words from documents or bag-of-words model

Syntax

```
newDocuments = removeLongWords(documents,len)
newBag = removeLongWords(bag,len)
```

Description

`newDocuments = removeLongWords(documents,len)` removes words of length `len` or greater from `documents`.

`newBag = removeLongWords(bag,len)` removes words of length `len` or greater from the `bagOfWords` object `bag`.

Examples

Remove Long Words from Document

Remove the words with seven or greater characters from a document.

```
document = tokenizedDocument("An example of a short sentence");
newDocument = removeLongWords(document,7)
```

```
newDocument = 
  tokenizedDocument:

   4 tokens: An of a short
```

Remove Long Words from Bag-of-Words Model

Remove the words with seven or greater characters from a bag-of-words model.

```
documents = tokenizedDocument([ ...
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents);
newBag = removeLongWords(bag,7)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [2x5 double]
      Vocabulary: ["an"    "of"    "a"    "short"    "second"]
        NumWords: 5
    NumDocuments: 2
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`bag` — Input bag-of-words model
`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` object.

`len` — Minimum length of words to remove
positive integer

Minimum length of words to remove, specified as a positive integer. The function removes words with `len` or greater characters.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

`newBag` — Output bag-of-words model
`bagOfWords` object

Output bag-of-words model, returned as a `bagOfWords` object.

See Also

`bagOfNgrams` | `bagOfWords` | `normalizeWords` | `removeShortWords` | `removeStopWords` | `removeWords` | `stopWords` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

removeNgrams

Remove n-grams from bag-of-n-grams model

Syntax

```
newBag = removeNgrams(bag,ngrams)
newBag = removeNgrams(bag,ngrams,'IgnoreCase',true)
newBag = removeNgrams(bag,idx)
```

Description

`newBag = removeNgrams(bag,ngrams)` removes the specified n-grams from the bag-of-n-grams model `bag`. The function, by default, is case sensitive.

`newBag = removeNgrams(bag,ngrams,'IgnoreCase',true)` removes n-grams ignoring case.

`newBag = removeNgrams(bag,idx)` specifies n-grams by numeric or logical indices in `bag.Ngrams`. This syntax is the same as `newBag = removeNgrams(bag,bag.Ngrams(idx,:))`.

Examples

Remove N-Grams from Bag-of-N-Grams Model

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create bag-of-n-grams model.

```
bag = bagOfNgrams(documents)
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154×8799 double]
      Vocabulary: [1×3092 string]
          Ngrams: [8799×2 string]
    NgramLengths: 2
       NumNgrams: 8799
    NumDocuments: 154
```

View the top five n-grams.

```
topkngrams(bag,5)
```

```
ans=5×3 table
         Ngram           Count    NgramLength
    ________________    _____    ___________

    "thou"    "art"      34           2
    "mine"    "eye"      15           2
    "thy"     "self"     14           2
    "thou"    "dost"     13           2
    "mine"    "own"      13           2
```

Remove the n-grams `["thou" "art"]` and `["thou" "dost"]` from the model. View the new top 5 n-grams.

```
ngrams = [...
    "thou" "art"
    "thou" "dost"];
bag = removeNgrams(bag,ngrams);
topkngrams(bag,5)
```

```
ans=5×3 table
          Ngram           Count    NgramLength
    _________________    _____    ___________

    "mine"    "eye"        15           2
    "thy"     "self"       14           2
    "mine"    "own"        13           2
    "thy"     "sweet"      12           2
    "thy"     "love"       11           2
```

Remove N-Grams from Bag-of-N-Grams Model by Index

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create bag-of-n-grams model.

```
bag = bagOfNgrams(documents)
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154x8799 double]
      Vocabulary: [1x3092 string]
          Ngrams: [8799x2 string]
    NgramLengths: 2
       NumNgrams: 8799
    NumDocuments: 154
```

View the first ten n-grams in the model.

```
bag.Ngrams(1:10,:)
```

```
ans = 10x2 string
    "fairest"      "creatures"
    "creatures"    "desire"
    "desire"       "increase"
    "increase"     "thereby"
    "thereby"      "beautys"
    "beautys"      "rose"
    "rose"         "might"
    "might"        "never"
    "never"        "die"
    "die"          "riper"
```

Remove the 9th and 10th n-grams from the model. View the new list of the first ten n-grams.

```
idx = [9 10];
bag = removeNgrams(bag,idx);
bag.Ngrams(1:10,:)
```

```
ans = 10x2 string
    "fairest"      "creatures"
    "creatures"    "desire"
    "desire"       "increase"
    "increase"     "thereby"
    "thereby"      "beautys"
    "beautys"      "rose"
    "rose"         "might"
    "might"        "never"
    "riper"        "time"
    "time"         "decease"
```

Input Arguments

bag — Input bag-of-n-grams model
`bagOfNgrams` object

Input bag-of-n-grams model, specified as a `bagOfNgrams` object.

`ngrams` — N-grams to remove
string array | character vector | cell array of character vectors

N-grams to remove, specified as a string array, character vector, or a cell array of character vectors.

If `ngrams` is a string array or cell array, then it has size `NumNgrams`-by-`maxN` , where `NumNgrams` is the number of n-grams, and `maxN` is the length of the largest n-gram. If `ngrams` is a character vector, then it represents a single word (unigram).

The value of `ngrams(i,j)` is the `j`th word of the `i`th n-gram. If the number of words in the `i`th n-gram is less than `maxN`, then the remaining entries of the `i`th row of `ngrams` are empty.

Example: `["An" ""; "An example"; "example" ""]`

Data Types: `string` | `char` | `cell`

`idx` — Indices of n-grams to remove
vector of numeric indices | vector of logical indices

Indices of n-grams to remove, specified as a vector of numeric indices or a vector of logical indices. The indices in `idx` correspond to the rows of the `bag.Ngrams`.

Example: `[1 5 10]`

See Also

`bagOfNgrams` | `bagOfWords` | `removeEmptyDocuments` | `removeInfrequentNgrams` | `removeWords` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2018a

removeShortWords

Remove short words from documents or bag-of-words model

Syntax

```
newDocuments = removeShortWords(documents,len)
newBag = removeShortWords(bag,len)
```

Description

`newDocuments = removeShortWords(documents,len)` removes words of length `len` or less from `documents`.

`newBag = removeShortWords(bag,len)` removes words of length `len` or less from the `bagOfWords` object `bag`.

Examples

Remove Short Words from Document

Remove the words with two or fewer characters from a document.

```
document = tokenizedDocument("An example of a short sentence");
newDocument = removeShortWords(document,2)
```

```
newDocument = 
  tokenizedDocument:

   3 tokens: example short sentence
```

Remove Short Words from Bag-of-Words Model

Remove the words with two or fewer characters from a bag-of-words model.

```
documents = tokenizedDocument([ ...
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents);
newBag = removeShortWords(bag,2)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [2x4 double]
      Vocabulary: ["example"    "short"    "sentence"    "second"]
        NumWords: 4
    NumDocuments: 2
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`bag` — Input bag-of-words model
`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` object.

`len` — Maximum length of words to remove
positive integer

Maximum length of words to remove, specified as a positive integer. The function removes words with `len` or fewer characters.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

`newBag` — Output bag-of-words model
`bagOfWords` object

Output bag-of-words model, returned as a `bagOfWords` object.

See Also

`bagOfNgrams` | `bagOfWords` | `normalizeWords` | `removeLongWords` | `removeWords` | `stopWords` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

removeStopWords

Remove stop words from documents

Syntax

```
newDocuments = removeStopWords(documents)
newDocuments = removeStopWords(documents,'IgnoreCase',false)
```

Description

Words like "a", "and", "to", and "the" (known as stop words) can add noise to data. Use this function to remove stop words before analysis.

The function supports English, Japanese, German, and Korean text. To learn how to use `removeStopWords` for other languages, see “Language Considerations” on page 1-313.

`newDocuments = removeStopWords(documents)` removes the stop words from the `tokenizedDocument` array `documents`. The function, by default, uses the stop word list given by the `stopWords` function according to the language details of `documents` and is case insensitive.

To remove a custom list of words, use the `removeWords` function.

`newDocuments = removeStopWords(documents,'IgnoreCase',false)` removes stop words with case matching the stop word list given by the `stopWords` function.

Tip Use `removeStopWords` before using the `normalizeWords` function as `removeStopWords` uses information that is removed by this function.

Examples

Remove Stop Words from Documents

Remove the stop words from an array of documents using `removeStopWords`. The `tokenizedDocument` function detects that the documents are in English, so `removeStopWords` removes English stop words.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
newDocuments = removeStopWords(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    3 tokens: example short sentence
    3 tokens: second short sentence
```

Remove Japanese Stop Words

Tokenize Japanese text using `tokenizedDocument`. The function automatically detects Japanese text.

```
str = [
    "ここは静かなので、とても穏やかです"
    "企業内の顧客データを利用し、今年の売り上げを調べることが出来た。"
    "私は先生です。私は英語を教えています。"];
documents = tokenizedDocument(str);
```

Remove stop words using `removeStopWords`. The function uses the language details from `documents` to determine which language stop words to remove.

```
documents = removeStopWords(documents)
```

```
documents = 
  3x1 tokenizedDocument:

     4 tokens: 静か 、 とても 穏やか
    10 tokens: 企業 顧客 データ 利用 、 今年 売り上げ 調べる 出来 。
     5 tokens: 先生 。 英語 教え 。
```

Remove German Stop Words from Documents

Tokenize German text using `tokenizedDocument`. The function automatically detects German text.

```
str = [
    "Guten Morgen. Wie geht es dir?"
    "Heute wird ein guter Tag."];
documents = tokenizedDocument(str)
```

```
documents = 
  2x1 tokenizedDocument:

    8 tokens: Guten Morgen . Wie geht es dir ?
    6 tokens: Heute wird ein guter Tag .
```

Remove stop words using the `removeStopWords` function. The function uses the language details from documents to determine which language stop words to remove.

```
documents = removeStopWords(documents)
```

```
documents = 
  2x1 tokenizedDocument:

    5 tokens: Guten Morgen . geht ?
    5 tokens: Heute wird guter Tag .
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

More About

Language Considerations

The `stopWords` and `removeStopWords` functions support English, Japanese, German, and Korean stop words only.

To remove stop words from other languages, use `removeWords` and specify your own stop words to remove.

Algorithms

Language Details

`tokenizedDocument` objects contain details about the tokens including language details. The language details of the input documents determine the behavior of `removeStopWords`. The `tokenizedDocument` function, by default, automatically detects the language of the input text. To specify the language details manually, use the `'Language'` name-value pair argument of `tokenizedDocument`. To view the token details, use the `tokenDetails` function.

See Also
`bagOfWords` | `normalizeWords` | `removeLongWords` | `removeShortWords` | `removeWords` | `stopWords` | `tokenizedDocument`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Language Considerations”
“Japanese Language Support”
“German Language Support”

Introduced in R2018b

removeWords

Remove selected words from documents or bag-of-words model

Syntax

```
newDocuments = removeWords(documents,words)
newBag = removeWords(bag,words)
newDocuments = removeWords( ___ ,'IgnoreCase',true)

newDocuments = removeWords(documents,idx)
newBag = removeWords(bag,idx)
```

Description

`newDocuments = removeWords(documents,words)` removes the specified words from `documents`. The function, by default, is case sensitive.

`newBag = removeWords(bag,words)` removes the specified words from the bag-of-words model `bag`. The function, by default, is case sensitive.

`newDocuments = removeWords( ___ ,'IgnoreCase',true)` removes words ignoring case using any of the previous syntaxes.

`newDocuments = removeWords(documents,idx)` removes words by specifying the numeric or logical indices `idx` of the words in `documents.Vocabulary`. This syntax is the same as `newDocuments = removeWords(documents,documents.Vocabulary(idx))`.

`newBag = removeWords(bag,idx)` removes words by specifying the numeric or logical indices `idx` of the words in `bag.Vocabulary`. This syntax is the same as `newBag = removeWords(bag,bag.Vocabulary(idx))`.

Examples

Remove Words from Documents

Remove words from an array of documents by inputting a string array of words to `removeWords`.

Create an array of tokenized documents.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
```

Remove the words "short" and "second".

```
words = ["short" "second"];
newDocuments = removeWords(documents,words)
```

```
newDocuments = 
  2x1 tokenizedDocument:
    5 tokens: an example of a sentence
    2 tokens: a sentence
```

Remove Custom List of Stop Words from Documents

To remove the default list of stop words using the language details of documents, use `removeStopWords`.

To remove a custom list of stop words, use the `removeWords` function. You can use the stop word list returned by the `stopWords` function as a starting point.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

View the first few documents.

```
documents(1:5)
```

```
ans = 
  5x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youth
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair th
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
```

Create a list of stop words starting with the output of the `stopWords` function.

```
customStopWords = [stopWords "thy" "thee" "thou" "dost" "doth"];
```

Remove the custom stop words from the documents and view the first few documents.

```
documents = removeWords(documents,customStopWords);
documents(1:5)
```

```
ans = 
  5x1 tokenizedDocument:

    62 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    61 tokens: forty winters shall besiege brow dig deep trenches beautys field youths proud live
    52 tokens: look glass tell face viewest time face form another whose fresh repair renewest be
    52 tokens: unthrifty loveliness why spend upon self beautys legacy natures bequest gives noth
    59 tokens: hours gentle work frame lovely gaze every eye dwell play tyrants same unfair fair
```

Remove Words from Documents by Index

Remove words from documents by inputting a vector of numeric indices to `removeWords`.

Create an array of tokenized documents.

```
documents = tokenizedDocument([
    "I love MATLAB"
    "I love MathWorks"])
```

```
documents = 
  2x1 tokenizedDocument:

    3 tokens: I love MATLAB
    3 tokens: I love MathWorks
```

View the vocabulary of `documents`.

```
documents.Vocabulary
```

```
ans = 1x4 string
    "I"    "love"    "MATLAB"    "MathWorks"
```

Remove the first and third words of the vocabulary from the documents by specifying the numeric indices `[1 3]`.

```
idx = [1 3];
newDocuments = removeWords(documents,idx)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    1 tokens: love
    2 tokens: love MathWorks
```

Alternatively, you can specify logical indices.

```
idx = logical([1 0 1 0]);
newDocuments = removeWords(documents,idx)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    1 tokens: love
    2 tokens: love MathWorks
```

Remove Stop Words from Bag-of-Words Model

Remove the stop words from a bag-of-words model by inputting a list of stop words to `removeWords`. Stop words are words such as "a", "the", and "in" which are commonly removed from text before analysis.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
bag = bagOfWords(documents);
newBag = removeWords(bag,stopWords)
```

```
newBag =
  bagOfWords with properties:

          Counts: [2x4 double]
      Vocabulary: ["example"    "short"    "sentence"    "second"]
        NumWords: 4
    NumDocuments: 2
```

Remove Words from Bag-of-Words Model by Index

Remove words from a bag-of-words model by inputting a vector of numeric indices to `removeWords`.

Create an array of tokenized documents.

```
documents = tokenizedDocument([
    "I love MATLAB"
    "I love MathWorks"]);
bag = bagOfWords(documents)
```

```
bag =
  bagOfWords with properties:

          Counts: [2x4 double]
      Vocabulary: ["I"    "love"    "MATLAB"    "MathWorks"]
        NumWords: 4
    NumDocuments: 2
```

View the vocabulary of `bag`.

```
bag.Vocabulary
```

```
ans = 1x4 string
    "I"    "love"    "MATLAB"    "MathWorks"
```

Remove the first and third words of the vocabulary from the bag-of-words model by specifying the numeric indices `[1 3]`.

```
idx = [1 3];
newBag = removeWords(bag,idx)
```

```
newBag =
  bagOfWords with properties:
          Counts: [2x2 double]
      Vocabulary: ["love"    "MathWorks"]
        NumWords: 2
    NumDocuments: 2
```

Alternatively, you can specify logical indices.

```
idx = logical([1 0 1 0]);
newBag = removeWords(bag,idx)
```

```
newBag = 
  bagOfWords with properties:

          Counts: [2x2 double]
      Vocabulary: ["love"    "MathWorks"]
        NumWords: 2
    NumDocuments: 2
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`bag` — Input bag-of-words model
`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` object.

`words` — Words to remove
string vector | character vector | cell array of character vectors

Words to remove, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats it as a single word.

Data Types: `string` | `char` | `cell`

`idx` — Indices of words in vocabulary to remove
vector of numeric indices | vector of logical indices

Indices of words to remove, specified as a vector of numeric indices or a vector of logical indices. The indices in `idx` correspond to the locations of the words in the `Vocabulary` property of the input documents or bag-of-words model.

Example: `[1 5 10]`

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

`newBag` — Output bag-of-words model
`bagOfWords` object

Output bag-of-words model, returned as a `bagOfWords` object.

See Also
`bagOfNgrams` | `bagOfWords` | `normalizeWords` | `removeEmptyDocuments` | `removeInfrequentWords` | `removeLongWords` | `removeNgrams` | `removeShortWords` | `stopWords` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2017b

replace

Replace substrings in documents

Syntax

`newDocuments = replace(documents,old,new)`

Description

`newDocuments = replace(documents,old,new)` replaces all occurrences of the substring or pattern `old` in `documents` with `new`.

Tip Use the `replace` function to replace substrings of the words in documents by specifying substrings or patterns. To replace entire words and n-grams in documents, use the `replaceWords` and `replaceNgrams` functions respectively.

Examples

Replace Substrings in Documents

Replace words in a document array.

```
documents = tokenizedDocument([
    "an extreme example"
    "another extreme example"])
```

```
documents = 
  2x1 tokenizedDocument:

    3 tokens: an extreme example
    3 tokens: another extreme example
```

```
newDocuments = replace(documents,"example","sentence")
```

```
newDocuments = 
  2x1 tokenizedDocument:

    3 tokens: an extreme sentence
    3 tokens: another extreme sentence
```

Replace substrings of the words.

```
newDocuments = replace(documents,"ex","X-")
```

```
newDocuments = 
  2x1 tokenizedDocument:

    3 tokens: an X-treme X-ample
    3 tokens: another X-treme X-ample
```

Replace Substrings in Documents Using Patterns

Remove digits from a document using a digits pattern.

Create an array of tokenized documents.

```
textData = [
    "Text Analytics Toolbox provides over 50 functions to analyze text data."
    "The bm25Similarity function measures document similarity."];
documents = tokenizedDocument(textData);
```

Replace instances of consecutive digits with the token `"<NUMBER>"` using the `replace` function. Specify a digits pattern using the `digitsPattern` function.

```
pat = digitsPattern;
newDocuments = replace(documents,pat,"<NUMBER>")
```

```
newDocuments = 
  2x1 tokenizedDocument:

    12 tokens: Text Analytics Toolbox provides over <NUMBER> functions to analyze text data .
     7 tokens: The bm<NUMBER>Similarity function measures document similarity .
```

Notice that the function replaces the digits in the token `"bm25Similarity"`.

To replace tokens consisting entirely of digits, use the `replace` function and specify a pattern that also includes text boundaries. Specify text boundaries using the `textBoundary` function.

```
pat = textBoundary + digitsPattern + textBoundary;
newDocuments = replace(documents,pat,"<NUMBER>")
```

```
newDocuments = 
  2x1 tokenizedDocument:

    12 tokens: Text Analytics Toolbox provides over <NUMBER> functions to analyze text data .
     7 tokens: The bm25Similarity function measures document similarity .
```

In this case, the function does not replace the digits in the token `"bm25Similarity"`.

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`old` — Substring or pattern to replace
string array | character vector | cell array of character vectors | `pattern` array

Substring or pattern to replace, specified as one of the following:

- String array
- Character vector
- Cell array of character vectors
- `pattern` array

`new` — New substring
string array | character vector | cell array of character vectors

New substring, specified as a string array, character vector, or cell array of character vectors.

Data Types: `string` | `char` | `cell`

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

See Also

`bagOfWords` | `decodeHTMLEntities` | `normalizeWords` | `regexprep` | `replaceNgrams` | `replaceWords` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

replaceWords

Replace words in documents

Syntax

```
newDocuments = replaceWords(documents,oldWords,newWords)
newDocuments = replaceWords(documents,oldWords,newWords,'IgnoreCase',true)
```

Description

`newDocuments = replaceWords(documents,oldWords,newWords)` updates the specified documents by replacing the words in `oldWords` with the corresponding words in `newWords`. The function, by default, is case sensitive.

`newDocuments = replaceWords(documents,oldWords,newWords,'IgnoreCase',true)` replaces the words in `oldWords` ignoring case.

Examples

Replace Words in Documents

Use the `replaceWords` function to replace shorthand words with their corresponding full words.

Create an array of tokenized documents.

```
str = [ ...
    "Increased activity Mon to Fri."
    "Reduced activity Sat to Sun."];
documents = tokenizedDocument(str)
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: Increased activity Mon to Fri .
    6 tokens: Reduced activity Sat to Sun .
```

Replace the shorthand words with their corresponding full words.

```
oldWords = ["Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun"];
newWords = ["Monday" "Tuesday" "Wednesday" "Thursday" "Friday" "Saturday" "Sunday"];
documents = replaceWords(documents,oldWords,newWords)
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: Increased activity Monday to Friday .
    6 tokens: Reduced activity Saturday to Sunday .
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`oldWords` — Words to replace
string array | character vector | cell array of character vectors

Words to replace, specified as a string array, character vector, or cell array of character vectors.

Data Types: `string` | `char` | `cell`

`newWords` — New words
string array | character vector | cell array of character vectors

New words, specified as a string array, character vector, or cell array of character vectors.

`newWords` must contain one word or be the same size as `oldWords`. If `newWords` contains only one word, then the function replaces all the words in `oldWords` with this word.

Data Types: `string` | `char` | `cell`

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

Tips

- To replace words in documents by specifying `pattern` arrays, use the `replace` function.

See Also

`decodeHTMLEntities` | `normalizeWords` | `replaceNgrams` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2019a

replaceNgrams

Replace n-grams in documents

Syntax

```
newDocuments = replaceNgrams(documents,oldNgrams,newNgrams)
newDocuments = replaceNgrams(documents,oldNgrams,newNgrams,'IgnoreCase',true)
```

Description

`newDocuments = replaceNgrams(documents,oldNgrams,newNgrams)` updates the specified documents by replacing the n-grams `oldNgrams` with the corresponding n-grams in `newNgrams`. The function, by default, is case sensitive.

`newDocuments = replaceNgrams(documents,oldNgrams,newNgrams,'IgnoreCase',true)` replaces the n-grams `oldNgrams` ignoring case.

Examples

Replace N-grams In Documents

Use the `replaceNgrams` function to replace abbreviations with their corresponding expanded forms.

Create an array of tokenized documents.

```
str = [ ...
    "Currently in Cambridge, MA."
    "Next stop, NY!"];
documents = tokenizedDocument(str)
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: Currently in Cambridge , MA .
    5 tokens: Next stop , NY !
```

Replace the tokens `"MA"` and `"NY"` with `"Massachusetts"` and `["New" "York"]` respectively. If the n-grams have different lengths, you must pad the rows with the empty string `""`. In this case, you must pad `"Massachusetts"` with a single empty string `""`.

```
oldNgrams = [
    "MA"
    "NY"];
newNgrams = [
    "Massachusetts" ""
    "New" "York"];
documents = replaceNgrams(documents,oldNgrams,newNgrams)
```

```
documents = 
  2x1 tokenizedDocument:
    6 tokens: Currently in Cambridge , Massachusetts .
    6 tokens: Next stop , New York !
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`oldNgrams` — N-grams to replace
string array | character vector | cell array of character vectors

N-grams to replace, specified as a string array, character vector, or a cell array of character vectors.

If `oldNgrams` is a string array or cell array, then it has size `NumNgrams`-by-`maxN` , where `NumNgrams` is the number of n-grams, and `maxN` is the length of the largest n-gram. If `oldNgrams` is a character vector, then it represents a single word (unigram).

The value of `oldNgrams(i,j)` is the `j`th word of the `i`th n-gram. If the number of words in the `i`th n-gram is less than `maxN`, then the remaining entries of the `i`th row of `oldNgrams` must be padded with the empty string `""`.

For example, to specify both the unigram `"Massachusetts"`, and the bigram `["New" "York"]`, specify the 2-by-2 string array `["Massachusetts" ""; "New" "York"]`, where `"Massachusetts"` is padded with a single empty string `""`.

Data Types: `string` | `char` | `cell`

`newNgrams` — New n-grams
string array | character vector | cell array of character vectors

New n-grams, specified as a string array, character vector, or a cell array of character vectors.

If `newNgrams` is a string array or cell array, then it has size `NumNgrams`-by-`maxN` , where `NumNgrams` is the number of n-grams, and `maxN` is the length of the largest n-gram. If `newNgrams` is a character vector, then it represents a single word (unigram).

The value of `newNgrams(i,j)` is the `j`th word of the `i`th n-gram. If the number of words in the `i`th n-gram is less than `maxN`, then the remaining entries of the `i`th row of `newNgrams` are empty.

`newNgrams` must have one row, or the same number of rows as `oldNgrams`.

For example, to specify both the unigram `"Massachusetts"`, and the bigram `["New" "York"]`, specify the 2-by-2 string array `["Massachusetts" ""; "New" "York"]`, where `"Massachusetts"` is padded with a single empty string `""`.

Data Types: `string` | `char` | `cell`

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

See Also

`decodeHTMLEntities` | `normalizeWords` | `removeWords` | `replaceWords` | `tokenizedDocument`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2019a

resume

Resume fitting LDA model

Syntax

```
updatedMdl = resume(ldaMdl,bag)
updatedMdl = resume(ldaMdl,counts)
updatedMdl = resume(___,Name,Value)
```

Description

`updatedMdl = resume(ldaMdl,bag)` returns an updated LDA model by training for more iterations on the bag-of-words or bag-of-n-grams model `bag`. The input `bag` must be the same model used to fit `ldaMdl`.

`updatedMdl = resume(ldaMdl,counts)` returns an updated LDA model by training for more iterations on the documents represented by the matrix of word counts `counts`. The input `counts` must be the same matrix used to fit `ldaMdl`.

`updatedMdl = resume(___,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Resume Fitting of LDA Model

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
```

```
    NumDocuments: 154
```

Fit an LDA model with four topics. The `resume` function does not support the default solver for `fitlda`. Set the LDA solver to be collapsed variational Bayes, zeroth order.

```
numTopics = 4;
mdl = fitlda(bag,numTopics,'Solver','cvb0')
```

```
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.00 |            |  3.292e+03 |         1.000 |             0 |
|          1 |       0.01 | 1.4970e-01 |  1.147e+03 |         1.000 |             0 |
|          2 |       0.00 | 7.1229e-03 |  1.091e+03 |         1.000 |             0 |
|          3 |       0.00 | 8.1261e-03 |  1.031e+03 |         1.000 |             0 |
|          4 |       0.00 | 8.8626e-03 |  9.703e+02 |         1.000 |             0 |
|          5 |       0.00 | 8.5486e-03 |  9.154e+02 |         1.000 |             0 |
|          6 |       0.00 | 7.4632e-03 |  8.703e+02 |         1.000 |             0 |
|          7 |       0.00 | 6.0480e-03 |  8.356e+02 |         1.000 |             0 |
|          8 |       0.00 | 4.5955e-03 |  8.102e+02 |         1.000 |             0 |
|          9 |       0.00 | 3.4068e-03 |  7.920e+02 |         1.000 |             0 |
|         10 |       0.00 | 2.5353e-03 |  7.788e+02 |         1.000 |             0 |
|         11 |       0.01 | 1.9089e-03 |  7.690e+02 |         1.222 |            10 |
|         12 |       0.01 | 1.2486e-03 |  7.626e+02 |         1.176 |             7 |
|         13 |       0.01 | 1.1243e-03 |  7.570e+02 |         1.125 |             7 |
|         14 |       0.00 | 9.1253e-04 |  7.524e+02 |         1.079 |             7 |
|         15 |       0.01 | 7.5878e-04 |  7.486e+02 |         1.039 |             6 |
|         16 |       0.00 | 6.6181e-04 |  7.454e+02 |         1.004 |             6 |
|         17 |       0.00 | 6.0400e-04 |  7.424e+02 |         0.974 |             6 |
|         18 |       0.00 | 5.6244e-04 |  7.396e+02 |         0.948 |             6 |
|         19 |       0.00 | 5.0548e-04 |  7.372e+02 |         0.926 |             5 |
|         20 |       0.00 | 4.2796e-04 |  7.351e+02 |         0.905 |             5 |
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|         21 |       0.00 | 3.4941e-04 |  7.334e+02 |         0.887 |             5 |
|         22 |       0.00 | 2.9495e-04 |  7.320e+02 |         0.871 |             5 |
|         23 |       0.00 | 2.6300e-04 |  7.307e+02 |         0.857 |             5 |
|         24 |       0.00 | 2.5200e-04 |  7.295e+02 |         0.844 |             4 |
|         25 |       0.00 | 2.4150e-04 |  7.283e+02 |         0.833 |             4 |
|         26 |       0.00 | 2.0549e-04 |  7.273e+02 |         0.823 |             4 |
|         27 |       0.00 | 1.6441e-04 |  7.266e+02 |         0.813 |             4 |
|         28 |       0.00 | 1.3256e-04 |  7.259e+02 |         0.805 |             4 |
|         29 |       0.00 | 1.1094e-04 |  7.254e+02 |         0.798 |             4 |
|         30 |       0.00 | 9.2849e-05 |  7.249e+02 |         0.791 |             4 |
=====================================================================================

mdl = 
  ldaModel with properties:

                     NumTopics: 4
             WordConcentration: 1
            TopicConcentration: 0.7908
```

```
      CorpusTopicProbabilities: [0.2654 0.2531 0.2480 0.2336]
    DocumentTopicProbabilities: [154x4 double]
        TopicWordProbabilities: [3092x4 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

View information about the fit.

```
mdl.FitInfo
```

```
ans = struct with fields:
          TerminationCode: 1
        TerminationStatus: "Relative tolerance on log-likelihood satisfied."
            NumIterations: 30
    NegativeLogLikelihood: 6.3042e+04
               Perplexity: 724.9445
                   Solver: "cvb0"
                  History: [1x1 struct]
```

Resume fitting the LDA model with a lower log-likelihood tolerance.

```
tolerance = 1e-5;
updatedMdl = resume(mdl,bag, ...
    'LogLikelihoodTolerance',tolerance)
```

```
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|         30 |       0.00 |            |  7.249e+02 |         0.791 |             0 |
|         31 |       0.01 | 8.0569e-05 |  7.246e+02 |         0.785 |             3 |
|         32 |       0.00 | 7.4692e-05 |  7.242e+02 |         0.779 |             3 |
|         33 |       0.00 | 6.9802e-05 |  7.239e+02 |         0.774 |             3 |
|         34 |       0.00 | 6.1154e-05 |  7.236e+02 |         0.770 |             3 |
|         35 |       0.00 | 5.3163e-05 |  7.233e+02 |         0.766 |             3 |
|         36 |       0.00 | 4.7807e-05 |  7.231e+02 |         0.762 |             3 |
|         37 |       0.00 | 4.1820e-05 |  7.229e+02 |         0.759 |             3 |
|         38 |       0.00 | 3.6237e-05 |  7.227e+02 |         0.756 |             3 |
|         39 |       0.00 | 3.1819e-05 |  7.226e+02 |         0.754 |             2 |
|         40 |       0.00 | 2.7772e-05 |  7.224e+02 |         0.751 |             2 |
|         41 |       0.00 | 2.5238e-05 |  7.223e+02 |         0.749 |             2 |
|         42 |       0.00 | 2.2052e-05 |  7.222e+02 |         0.747 |             2 |
|         43 |       0.00 | 1.8471e-05 |  7.221e+02 |         0.745 |             2 |
|         44 |       0.00 | 1.5638e-05 |  7.221e+02 |         0.744 |             2 |
|         45 |       0.00 | 1.3735e-05 |  7.220e+02 |         0.742 |             2 |
|         46 |       0.00 | 1.2298e-05 |  7.219e+02 |         0.741 |             2 |
|         47 |       0.00 | 1.0905e-05 |  7.219e+02 |         0.739 |             2 |
|         48 |       0.00 | 9.5581e-06 |  7.218e+02 |         0.738 |             2 |
=====================================================================================
```

```
updatedMdl = 
  ldaModel with properties:

                     NumTopics: 4
             WordConcentration: 1
```

```
            TopicConcentration: 0.7383
      CorpusTopicProbabilities: [0.2679 0.2517 0.2495 0.2309]
    DocumentTopicProbabilities: [154x4 double]
        TopicWordProbabilities: [3092x4 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

View information about the fit.

```
updatedMdl.FitInfo
```

```
ans = struct with fields:
          TerminationCode: 1
        TerminationStatus: "Relative tolerance on log-likelihood satisfied."
            NumIterations: 48
    NegativeLogLikelihood: 6.3001e+04
               Perplexity: 721.8357
                   Solver: "cvb0"
                  History: [1x1 struct]
```

Input Arguments

`ldaMdl` — Input LDA model
`ldaModel` object

Input LDA model, specified as an `ldaModel` object. To resume fitting a model, you must fit `ldaMdl` with solver `'savb'`, `'avb'`, or `'cvb0'`.

`bag` — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`counts` — Frequency counts of words
matrix of nonnegative integers

Frequency counts of words, specified as a matrix of nonnegative integers. If you specify `'DocumentsIn'` to be `'rows'`, then the value `counts(i,j)` corresponds to the number of times the *j*th word of the vocabulary appears in the *i*th document. Otherwise, the value `counts(i,j)` corresponds to the number of times the *i*th word of the vocabulary appears in the *j*th document.

Note The arguments `bag` and `counts` must be the same used to fit `ldaMdl`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'LogLikelihoodTolerance',0.001` specifies a log-likelihood tolerance of 0.001.

Solver Options

`DocumentsIn` — Orientation of documents
`'rows'` (default) | `'columns'`

Orientation of documents in the word count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Input is a matrix of word counts with rows corresponding to documents.
- `'columns'` – Input is a transposed matrix of word counts with columns corresponding to documents.

This option only applies if you specify the input documents as a matrix of word counts.

Note If you orient your word count matrix so that documents correspond to columns and specify `'DocumentsIn','columns'`, then you might experience a significant reduction in optimization-execution time.

`FitTopicConcentration` — Option for fitting topic concentration parameter
`true` | `false`

Option for fitting topic concentration, specified as the comma-separated pair consisting of `'FitTopicConcentration'` and either `true` or `false`.

The default value is the value used to fit `ldaMdl`.

Example: `'FitTopicConcentration',true`

Data Types: `logical`

`FitTopicProbabilities` — Option for fitting topic probabilities
`true` | `false`

Option for fitting topic concentration, specified as the comma-separated pair consisting of `'FitTopicConcentration'` and either `true` or `false`.

The default value is the value used to fit `ldaMdl`.

The function fits the Dirichlet prior $\alpha = \alpha_0 (p_1\ p_2\ \cdots\ p_K)$ on the topic mixtures, where α_0 is the topic concentration and $p_1, \ldots, p_K$ are the corpus topic probabilities which sum to 1.

Example: `'FitTopicProbabilities',true`

Data Types: `logical`

`LogLikelihoodTolerance` — Relative tolerance on log-likelihood
`0.0001` (default) | positive scalar

Relative tolerance on log-likelihood, specified as the comma-separated pair consisting of `'LogLikelihoodTolerance'` and a positive scalar. The optimization terminates when this tolerance is reached.

Example: `'LogLikelihoodTolerance',0.001`

Batch Solver Options

`IterationLimit` — Maximum number of iterations
`100` (default) | positive integer

Maximum number of iterations, specified as the comma-separated pair consisting of `'IterationLimit'` and a positive integer.

This option supports models fitted with batch solvers only (`'cgs'`, `'avb'`, and `'cvb0'`).

Example: `'IterationLimit',200`

Stochastic Solver Options

`DataPassLimit` — Maximum number of passes through data
1 (default) | positive integer

Maximum number of passes through the data, specified as the comma-separated pair consisting of `'DataPassLimit'` and a positive integer.

If you specify `'DataPassLimit'` but not `'MiniBatchLimit'`, then the default value of `'MiniBatchLimit'` is ignored. If you specify both `'DataPassLimit'` and `'MiniBatchLimit'`, then `resume` uses the argument that results in processing the fewest observations.

This option supports models fitted with stochastic solvers only (`'savb'`).

Example: `'DataPassLimit',2`

`MiniBatchLimit` — Maximum number of mini-batch passes
positive integer

Maximum number of mini-batch passes, specified as the comma-separated pair consisting of `'MiniBatchLimit'` and a positive integer.

If you specify `'MiniBatchLimit'` but not `'DataPassLimit'`, then `resume` ignores the default value of `'DataPassLimit'`. If you specify both `'MiniBatchLimit'` and `'DataPassLimit'`, then `resume` uses the argument that results in processing the fewest observations. The default value is `ceil(numDocuments/MiniBatchSize)`, where `numDocuments` is the number of input documents.

This option supports models fitted with stochastic solvers only (`'savb'`).

Example: `'MiniBatchLimit',200`

`MiniBatchSize` — Mini-batch size
1000 (default) | positive integer

Mini-batch size, specified as the comma-separated pair consisting of `'MiniBatchLimit'` and a positive integer. The function processes `MiniBatchSize` documents in each iteration.

This option supports models fitted with stochastic solvers only (`'savb'`).

Example: `'MiniBatchSize',512`

Display Options

`ValidationData` — Validation data
`[]` (default) | `bagOfWords` object | `bagOfNgrams` object | sparse matrix of word counts

Validation data to monitor optimization convergence, specified as the comma-separated pair consisting of `'ValidationData'` and a `bagOfWords` object, a `bagOfNgrams` object, or a sparse matrix of word counts. If the validation data is a matrix, then the data must have the same orientation and the same number of words as the input documents.

`ValidationFrequency` — Frequency of model validation
positive integer

Frequency of model validation in number of iterations, specified as the comma-separated pair consisting of `'ValidationFrequency'` and a positive integer.

The default value depends on the solver used to fit the model. For the stochastic solver, the default value is 10. For the other solvers, the default value is 1.

`Verbose` — Verbosity level
1 (default) | 0

Verbosity level, specified as the comma-separated pair consisting of `'Verbose'` and one of the following:

- 0 – Do not display verbose output.
- 1 – Display progress information.

Example: `'Verbose',0`

Output Arguments

`updatedMdl` — Updated LDA model
`ldaModel` object (default)

Updated LDA model, returned as an `ldaModel` object.

See Also
`bagOfNgrams` | `bagOfWords` | `fitlda` | `ldaModel` | `logp` | `predict` | `transform` | `wordcloud`

Topics
“Analyze Text Data Using Topic Models”
“Choose Number of Topics for LDA Model”
“Compare LDA Solvers”
“Analyze Text Data Using Multiword Phrases”
“Classify Text Data Using Deep Learning”

Introduced in R2017b

rougeEvaluationScore

Evaluate translation or summarization with ROUGE similarity score

Syntax

```
score = rougeEvaluationScore(candidate,references)
score = rougeEvaluationScore(candidate,references,Name,Value)
```

Description

The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) scoring algorithm evaluates the similarity between a candidate document and a collection of reference documents. Use the ROUGE score to evaluate the quality of document translation and summarization models.

`score = rougeEvaluationScore(candidate,references)` returns the ROUGE score between the specified candidate document and the reference documents. The function, by default, computes unigram overlaps between `candidate` and `references`. This is also known as the ROUGE-N metric with n-gram length 1. For more information, see "ROUGE Score" on page 1-338.

`score = rougeEvaluationScore(candidate,references,Name,Value)` specifies additional options using one or more name-value pairs.

Examples

Evaluate Similarity

Specify the candidate document as a `tokenizedDocument` object.

```
str = "the fast brown fox jumped over the lazy dog";
candidate = tokenizedDocument(str)
```

```
candidate = 
  tokenizedDocument:

   9 tokens: the fast brown fox jumped over the lazy dog
```

Specify the reference documents as a `tokenizedDocument` array.

```
str = [
    "the quick brown animal jumped over the lazy dog"
    "the quick brown fox jumped over the lazy dog"];
references = tokenizedDocument(str)
```

```
references = 
  2x1 tokenizedDocument:

    9 tokens: the quick brown animal jumped over the lazy dog
    9 tokens: the quick brown fox jumped over the lazy dog
```

Calculate the ROUGE score between the candidate document and the reference documents.

```
score = rougeEvaluationScore(candidate,references)
```

```
score = 0.8889
```

Specify N-Gram Lengths

Specify the candidate document as a `tokenizedDocument` object.

```
str = "a simple summary document containing some words";
candidate = tokenizedDocument(str)
```

```
candidate = 
  tokenizedDocument:

   7 tokens: a simple summary document containing some words
```

Specify the reference documents as a `tokenizedDocument` array.

```
str = [
    "a simple document"
    "another document with some words"];
references = tokenizedDocument(str)
```

```
references = 
  2x1 tokenizedDocument:

    3 tokens: a simple document
    5 tokens: another document with some words
```

Calculate the ROUGE score between the candidate document and the reference documents using the default options.

```
score = rougeEvaluationScore(candidate,references)
```

```
score = 1
```

The `rougeEvaluationScore` function, by default, compares unigram (single-token) overlaps between the candidate document and the reference documents. Because the ROUGE score is a recall-based measure, if one of the reference documents is made up entirely of unigrams that appear in the candidate document, the resulting ROUGE score is one. In this scenario, the output of the `rougeEvaluationScore` function is uninformative.

For a more meaningful result, calcualte the ROUGE score again using bigrams by setting the `'NgramLength'` option to `2`. The resulting score is less than one, since every reference document contain bigrams that do not appear in the candidate document.

```
score = rougeEvaluationScore(candidate,references,'NgramLength',2)
```

```
score = 0.5000
```

Input Arguments

`candidate` — Candidate document

`tokenizedDocument` scalar | string array | cell array of character vectors

Candidate document, specified as a `tokenizedDocument` scalar, a string array, or a cell array of character vectors. If `candidate` is not a `tokenizedDocument` scalar, then it must be a row vector representing a single document, where each element is a word.

`references` — Reference documents

`tokenizedDocument` array | string array | cell array of character vectors

Reference documents, specified as a `tokenizedDocument` array, a string array, or a cell array of character vectors. If `references` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To evaluate against multiple reference documents, use a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `scores = rougeEvaluationScore(candidate,references,'ROUGEMethod','weighted-subsequences')` specifies to use the weighted subsequences ROUGE method.

`ROUGEMethod` — ROUGE method

`'n-grams'` (default) | `'longest-common-subsequences'` | `'weighted-subsequences'` | `'skip-bigrams'` | `'skip-bigrams-and-unigrams'`

ROUGE method, specified as the comma-separated pair consisting of `'ROUGEMethod'` and one of the following:

- `'n-grams'` – Evaluate the ROUGE score using n-gram overlaps between the candidate document and the reference documents. This is also known as the ROUGE-N metric.
- `'longest-common-subsequences'` – Evaluate the ROUGE score using Longest Common Subsequence (LCS) statistics. This is also known as the ROUGE-L metric.
- `'weighted-subsequences'` – Evaluate the ROUGE score using weighted longest common subsequence statistics. This method favors consecutive LCSs. This is also known as the ROUGE-W metric.
- `'skip-bigrams'` – Evaluate the ROUGE score using skip-bigram (any pair of words in sentence order) co-occurrence statistics. This is also known as the ROUGE-S metric.
- `'skip-bigrams-and-unigrams'` – Evaluate the ROUGE score using skip-bigram and unigram co-occurrence statistics. This is also known as the ROUGE-SU metric.

`NgramLength` — N-gram length

1 (default) | positive integer

N-gram length used for the `'n-grams'` ROUGE method (ROUGE-N), specified as the comma-separated pair consisting of `'NgramLength'` and a positive integer.

If the `'ROUGEMethod'` option is not `'n-grams'`, then the `'NgramLength'` option has no effect.

Tip If the longest document in `references` has fewer than `NgramLength` words, then the resulting ROUGE score is `NaN`. If `candidate` has fewer than `NgramLength` words, then the resulting ROUGE score is zero. To ensure that `rougeEvaluationScore` returns nonzero scores for very short documents, set `NgramLength` to a positive integer smaller than the length of `candidate` and the length of the longest document in `references`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`SkipDistance` — Skip distance
4 (default) | positive integer

Skip distance used for the `'skip-bigrams'` and `'skip-bigrams-and-unigrams'` ROUGE methods (ROUGE-S and ROUGE-SU), specified as the comma-separated pair consisting of `'SkipDistance'` and a positive integer.

If the `'ROUGEMethod'` option is not `'skip-bigrams'` or `'skip-bigrams-and-unigrams'`, then the `'SkipDistance'` option has no effect.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — ROUGE score
scalar

ROUGE score, returned as a scalar value in the range [0,1] or `NaN`.

A ROUGE score close to zero indicates poor similarity between `candidate` and `references`. A ROUGE score close to one indicates strong similarity between `candidate` and `references`. If `candidate` is identical to one of the reference documents, then `score` is 1. If `candidate` and `references` are both empty documents, then the resulting ROUGE score is `NaN`.

Tip If the longest document in `references` has fewer than `NgramLength` words, then the resulting ROUGE score is `NaN`. If `candidate` has fewer than `NgramLength` words, then the resulting ROUGE score is zero. To ensure that `rougeEvaluationScore` returns nonzero scores for very short documents, set `NgramLength` to a positive integer smaller than the length of `candidate` and the length of the longest document in `references`.

Algorithms

ROUGE Score

The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) scoring algorithm [1] calculates the similarity between a candidate document and a collection of reference documents. Use the ROUGE score to evaluate the quality of document translation and summarization models.

N-gram Co-Occurrence Statistics (ROUGE-N)

Given an n-gram length *n*, the ROUGE-N metric between a candidate document and a *single* reference document is given by

$$\text{ROUGE-N}_{\text{single}}(\text{candidate, reference}) = \frac{\sum_{r_i \in \text{ reference}} \sum_{\text{n-gram} \in r_i} \text{Count(n-gram, candidate)}}{\sum_{r_i \in \text{ reference}} \text{numNgrams}(r_i)},$$

where the elements r_i are sentences in the reference document, Count(n-gram, candidate) is the number of times the specified n-gram occurs in the candidate document and *numNgrams(r_i)* is the number of n-grams in the specified reference sentence r_i.

For sets of multiple reference documents, the ROUGE-N metric is given by

$$\text{ROUGE-N(candidate, references)} = \text{m}ax_k\left\{\text{ROUGE-N}_{\text{single}}(\text{candidate, references}_k)\right\}.$$

To use the ROUGE-N metric, set the `'ROUGEMethod'` option to `'n-grams'`.

Longest Common Subsequence (ROUGE-L)

Given a sentence $d = [w_1, \ldots, w_m]$ and a sentence s, where the elements s_i correspond to words, the subsequence $[w_{i_1}, \ldots, w_{i_k}]$ is a *common subsequence* of d and s if $w_{i_j'} \in \{s_1, \ldots, s_n\}$ for $j = 1, \ldots, k$ and $i_1 < \cdots < i_k$, where the elements of s are the words of the sentence and k is the length of the subsequence. The subsequence $[w_{i_1}, \ldots, w_{i_k}]$ is a longest common subsequence (LCS) if the subsequence length k is maximal.

Given a candidate document and a single reference document the *union* of the longest common subsequences is given by

$$LCS_{\cup}(\text{candidate, reference}) = \bigcup_{r_i \in \text{ reference}} \{w \mid w \in \text{LCS}(\text{candidate, r}_i)\},$$

where LCS(candidate, r_i) is the set of longest common subsequences in the candidate document and the sentence r_i from a reference document.

The ROUGE-L metric is an F-score measure. To calculate it, first calculate the recall and precision scores given by

$$R_{\text{lcs}}(\text{candidate, reference}) = \frac{\sum_{r_i \in \text{ reference}} |\text{LCS}_{\cup}(\text{candidate,r}_i)|}{\text{numWords(reference)}}$$

$$P_{\text{lcs}}(\text{candidate, reference}) = \frac{\sum_{r_i \in \text{ reference}} |\text{LCS}_{\cup}(\text{candidate,r}_i)|}{\text{numWords(candidate)}}.$$

Then, the ROUGE-L metric between a candidate document and a *single* reference document is given by the F-score measure

$$\text{ROUGE-L}_{\text{single}}(\text{candidate, reference})$$
$$= \frac{(1+\beta^2)R_{\text{lcs}}(\text{candidate, reference})P_{\text{lcs}}(\text{candidate, reference})}{R_{\text{lcs}}(\text{candidate, reference}) + \beta^2 P_{\text{lcs}}(\text{candidate, reference})},$$

where the parameter β controls the relative importance of the precision and recall. Because the ROUGE score favors recall, β is typically set to a high value.

For sets of multiple reference documents, the ROUGE-L metric is given by

$$\text{ROUGE-L(candidate, references)} = \mathrm{m}ax_k\left\{\text{ROUGE-L}_{\text{single}}(\text{candidate, references}_k)\right\}.$$

To use the ROUGE-L metric, set the `'ROUGEMethod'` option to `'longest-common-subsequences'`.

Weighted Longest Common Subsequence (ROUGE-W)

Given a weighting function *f* such that *f* has the property *f(x+y)>f(x)+f(y)* for any positive integers *x* and *y*, define WLCS(candidate, reference) to be the length of the longest consecutive matches encountered in the candidate document and a single reference document scored by the weighting function *f*. For more information about calculating this value, see [1].

The ROUGE-W is metric given an F-score measure which requires the recall and precision scores given by

$$R_{\text{wlcs}}(\text{candidate, reference}) = f^{-1}\left(\frac{\text{WLCS(candidate, reference)}}{f(\text{numWords(reference)})}\right)$$

$$P_{\text{wlcs}}(\text{candidate, reference}) = f^{-1}\left(\frac{\text{WLCS(candidate, reference)}}{f(\text{numWords(candidate))}}\right).$$

The ROUGE-W metric between a candidate document and a *single* reference document is given by the F-score measure

$$\text{ROUGE-W}_{\text{single}}(\text{candidate, reference}) = \frac{(1+\beta^2)R_{\text{wlcs}}(\text{candidate, reference})P_{\text{wlcs}}(\text{candidate, reference})}{R_{\text{wlcs}}(\text{candidate, reference}) + \beta^2 P_{\text{wlcs}}(\text{candidate, reference})},$$

where the parameter β controls the relative importance of the precision and recall. Because the ROUGE score favors recall, β is typically set to a high value.

For multiple reference documents, the ROUGE-W metric is given by

$$\text{ROUGE-W(candidate, references)} = \mathrm{m}ax_k\left\{\text{ROUGE-W}_{\text{single}}(\text{candidate, references}_k)\right\}.$$

To use the ROUGE-W metric, set the `'ROUGEMethod'` option to `'weighted-longest-common-subsequences'`.

Skip-Bigram Co-Occurrence Statistics (ROUGE-S)

A *skip-bigram* is an ordered pair of words in a sentence allowing for arbitrary gaps between them. That is, given a sentence $c_i = [c_{i1}, \ldots, c_{im}]$ from a candidate document, where the elements c_{ij} correspond to the words in the sentence, the pair of words $[c_{ij_1'}, c_{ij_2''}]$ is a *skip-bigram* if $j_1' < j_2''$.

The ROUGE-S metric is an F-score measure. To calculate it, first calculate the recall and precision scores given by

$$R_{\text{skip2}}(\text{candidate, reference}) = \frac{\sum_{r_i \in \text{ reference}} \sum_{\text{skip-bigram} \in r_i} \text{Count(skip-bigram, candidate)}}{\sum_{r_i \in \text{ reference}} \text{numSkipBigrams}(r_i)}$$

$$P_{\text{skip2}}(\text{candidate},\ \text{reference}) = \frac{\sum_{r_i \in \text{reference}} \sum_{\text{skip-bigram} \in r_i} \text{Count}(\text{skip-bigram},\ \text{candidate})}{\sum_{c_i \in\ \text{candidate}} \text{numSkipBigrams}(c_i)}.$$

where the elements r_i and c_i are sentences in the reference document and candidate document, respectively, Count(skip-bigram, candidate) is the number of times the specified skip-bigram occurs in the candidate document, and *numSkipBigrams(s)* is the number of skip-bigrams in the sentence *s*.

Then, the ROUGE-S metric between a candidate document and a *single* reference document is given by the F-score measure

$$\text{ROUGE-S}_{\text{single}}(\text{candidate},\ \text{reference})$$
$$= \frac{(1+\beta^2)R_{\text{skip2}}(\text{candidate},\ \text{reference})P_{\text{skip2}}(\text{candidate},\ \text{reference})}{R_{\text{skip2}}(\text{candidate},\ \text{reference}) + \beta^2 P_{\text{skip2}}(\text{candidate},\ \text{reference})},$$

For sets of multiple reference documents, the ROUGE-S metric is given by

$$\text{ROUGE-S}(\text{candidate, references}) = \text{m}ax_k\left\{\text{ROUGE-S}_{\text{single}}(\text{candidate, references}_k)\right\}.$$

To use the ROUGE-S metric, set the `'ROUGEMethod'` option to `'skip-bigrams'`.

Skip-Bigram and Unigram Co-Occurrence Statistics (ROUGE-SU)

To also include unigram co-occurrence statistics in the ROUGE-S metric, introduce unigram counts into the recall and precision scores for ROUGE-S. This is equivalent to including start tokens in the candidate and reference documents, since

$$\sum_{\text{skip-bigram} \in r_i} (\text{Count}(\text{skip-bigram},\ \text{candidate})) + \sum_{\text{unigram} \in r_i} (\text{Count}(\text{unigram},\ \text{candidate})$$
$$= \sum_{\text{skip-bigram} \in r_i^+} \left(\text{Count}(\text{skip-bigram},\ \text{candidate}^+)\right),$$

where *Count(unigram,candidate)* is the number of times the specified unigram appears in the candidate document, and r_i^+ and candidate$^+$ denote the reference sentence and the candidate document augmented with start tokens, respectively.

For sets of multiple reference documents, the ROUGE-SU metric is given by

$$\text{ROUGE-SU}(\text{candidate, references}) = \text{m}ax_k\left\{\text{ROUGE-S}_{\text{single}}(\text{candidate}^+, \text{references}_k^+)\right\},$$

where reference$^+$ is the reference document with sentences augmented with start tokens.

To use the ROUGE-SU metric, set the `'ROUGEMethod'` option to `'skip-bigrams-and-unigrams'`.

References

[1] Lin, Chin-Yew. "Rouge: A package for automatic evaluation of summaries." In *Text Summarization Branches Out*, pp. 74-81. 2004.

See Also

`bleuEvaluationScore` | `bm25Similarity` | `cosineSimilarity` | `extractSummary` | `lexrankScores` | `mmrScores` | `textrankScores` | `tokenizedDocument`

Topics

“Sequence-to-Sequence Translation Using Attention”

Introduced in R2020a

splitGraphemes

Split string into graphemes

Syntax

```
newStr = splitGraphemes(str)
```

Description

`newStr = splitGraphemes(str)` splits the string `str` into graphemes. A grapheme (also known as grapheme cluster) is the Unicode term for human-perceived characters.

Examples

Split Text into Graphemes

Split text into graphemes using the `splitGraphemes` function.

A grapheme (also known as grapheme clusters) is the Unicode term for human-perceived characters. Some graphemes contain multiple code units. For example, the "smiling face with sunglasses" emoji (□□with code point U+1F60E) is a single grapheme but comprises two UTF16 code units `"D83D"` and `"DE0E"`.

Split the text `"Smile! □□"` into graphemes.

```
str = "Smile! " + compose("\xD83D\xDE0E")
```

```
str = 
"Smile! □□"
```

```
newStr = splitGraphemes(str)
```

```
newStr = 8x1 string
    "S"
    "m"
    "i"
    "l"
    "e"
    "!"
    " "
    "□□"
```

Here, the function does not split the emoji into multiple characters.

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors. For string array and cell array input, each element of `str` must have the same number of graphemes.

If the number of graphemes is not the same for every element of `str`, then call the function in a for-loop to split the elements of `str` one at a time.

Data Types: `string` | `char` | `cell`

Output Arguments

`newStr` — Split graphemes
string array | cell array of character vectors

Split graphemes, returned as a string array or a cell array of character vectors. If `str` is a string array, then `newStr` is also a string array. Otherwise, `newStr` is a cell array of character vectors.

The size of `newStr` depends on the input:

- If `str` is a string scalar or a character vector, then `newStr` is an `numGraphemes`-by-1 string array or cell array, where `numGraphemes` is the number of graphemes.
- If `str` is an `M`-by-1 string array or cell array, then `newStr` is a `M`-by-`numGraphemes` array.
- If `str` is a 1-by-`N` string array or cell array, then `newStr` is a 1-by-`N`-by-`numGraphemes` array.

For a string array or cell array of any size, the function orients the split graphemes along the first trailing dimension with size 1.

See Also

`editDistance` | `editDistanceSearcher` | `knnsearch` | `rangesearch` | `split` | `tokenizedDocument`

Topics
"Create Custom Spelling Correction Function Using Edit Distance Searchers"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"

Introduced in R2019a

splitSentences

Split text into sentences

Syntax

```
newStr = splitSentences(str)
```

Description

`newStr = splitSentences(str)` splits `str` into an array of sentences.

Examples

Split Text into Sentences

Read the text from the example file `sonnets.txt` and split it into sentences.

```
filename = "sonnets.txt";
str = extractFileText(filename);
sentences = splitSentences(str);
```

View the first few sentences.

```
sentences(1:10)
```

```
ans = 10x1 string
    "THE SONNETS"
    "by William Shakespeare"
    "I"
    "From fairest creatures we desire increase,..."
    "II"
    "When forty winters shall besiege thy brow,..."
    "How much more praise deserv'd thy beauty's use,..."
    "This were to be new made when thou art old,..."
    "III"
    "Look in thy glass and tell the face thou viewest..."
```

Input Arguments

`str` — Input text
string scalar | character vector | scalar cell array containing a character vector

Input text, specified as a string scalar, a character vector, or a scalar cell array containing a character vector.

Data Types: `string` | `char` | `cell`

Output Arguments

`newStr` — Output text
string array | character vector | cell array of character vectors

Output text, returned as a string array, a character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

Algorithms

If emoticons or emoji characters appear after a terminating punctuation character, then the function splits the sentence after the emoticons and emoji.

See Also

`addSentenceDetails` | `corpusLanguage` | `decodeHTMLEntities` | `erasePunctuation` | `eraseTags` | `eraseURLs` | `lower` | `tokenizedDocument` | `upper`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Language Considerations"

Introduced in R2018a

stopWords

List of stop words

Syntax

```
words = stopWords
words = stopWords('Language',language)
```

Description

Words like "a", "and", "to", and "the" (known as stop words) can add noise to data. Use stop word lists to help create custom lists of words to remove before analysis.

To remove the default list of stop words from tokenized documents using the language details of the documents, use `removeStopWords`. To remove a custom list of words from tokenized documents, use `removeWords`.

The function returns English, Japanese, German, and Korean stop word lists.

`words = stopWords` returns a string array of common English words which can be removed from documents before analysis.

`words = stopWords('Language',language)` specifies the stop word language.

Examples

Remove Custom List of Stop Words from Documents

To remove the default list of stop words using the language details of documents, use `removeStopWords`.

To remove a custom list of stop words, use the `removeWords` function. You can use the stop word list returned by the `stopWords` function as a starting point.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

View the first few documents.

```
documents(1:5)
```

```
ans = 
  5x1 tokenizedDocument:
    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youth
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair th
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
```

Create a list of stop words starting with the output of the `stopWords` function.

```
customStopWords = [stopWords "thy" "thee" "thou" "dost" "doth"];
```

Remove the custom stop words from the documents and view the first few documents.

```
documents = removeWords(documents,customStopWords);
documents(1:5)
```

```
ans = 
  5x1 tokenizedDocument:

    62 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    61 tokens: forty winters shall besiege brow dig deep trenches beautys field youths proud live
    52 tokens: look glass tell face viewest time face form another whose fresh repair renewest be
    52 tokens: unthrifty loveliness why spend upon self beautys legacy natures bequest gives noth
    59 tokens: hours gentle work frame lovely gaze every eye dwell play tyrants same unfair fairl
```

List of English Stop Words

Get a list of English stop words using the `stopWords` function. For readability, reshape the output.

```
words = stopWords;
reshape(words,[25 9])
```

```
ans = 25x9 string
  Columns 1 through 6

    "a"          "but"         "during"     "hows"       "it's"      "said"
    "about"      "by"          "each"       "however"    "it’s"      "says"
    "above"      "can"         "either"     "i"          "its"       "see"
    "across"     "can't"       "for"        "i'd"        "let's"     "she"
    "after"      "can’t"       "from"       "i’d"        "let’s"     "she'd"
    "all"        "cant"        "given"      "i'll"       "lets"      "she’d"
    "along"      "cannot"      "had"        "i’ll"       "may"       "shed"
    "also"       "could"       "has"        "i'm"        "me"        "she'll"
    "am"         "couldn't"    "have"       "i’m"        "more"      "she’ll"
    "an"         "couldn’t"    "having"     "im"         "most"      "shell"
    "and"        "couldnt"     "he"         "i've"       "much"      "should"
    "any"        "did"         "he'd"       "i’ve"       "must"      "since"
    "are"        "didn't"      "he’d"       "ive"        "my"        "so"
    "aren't"     "didn’t"      "hed"        "if"         "no"        "some"
    "aren’t"     "didnt"       "he'll"      "in"         "not"       "such"
    "arent"      "do"          "he’ll"      "instead"    "now"       "than"
    "as"         "does"        "her"        "into"       "of"        "that"
    "at"         "doesn't"     "here"       "is"         "on"        "the"
    "be"         "doesn’t"     "hers"       "isn't"      "one"       "their"
    "because"    "doesnt"     "him"        "isn’t"      "only"     "them"
    "been"       "doing"      "himself"    "isnt"       "or"       "then"
    "before"     "done"       "his"        "it"         "other"    "there"
    "being"      "don't"      "how"        "it'll"      "our"      "therefore"
    "between"    "don’t"      "how's"      "it’ll"      "out"      "these"
    "both"       "dont"       "how’s"      "itll"       "over"     "they"

  Columns 7 through 9

    "this"       "we’re"      "who’ve"
    "those"      "we've"      "whove"
    "through"    "we’ve"      "will"
    "to"         "weve"       "with"
    "too"        "were"       "within"
    "towards"    "what"       "without"
    "under"      "what's"     "won't"
    "until"      "what’s"     "won’t"
    "us"         "whats"      "would"
    "use"        "when"       "wouldn't"
    "used"       "when's"     "wouldn’t"
    "uses"       "when’s"     "you"
    "using"      "whens"      "you'd"
    "very"       "where"      "you’d"
    "want"       "whether"    "youd"
    "was"        "which"      "you'll"
    "wasn't"     "while"      "you’ll"
    "wasn’t"     "who"        "youll"
    "wasnt"      "who'll"     "you're"
    "we"         "who’ll"     "you’re"
    "we'd"       "wholl"      "youre"
    "we’d"       "who's"      "you've"
    "we'll"      "who’s"      "you’ve"
    "we’ll"      "whos"       "youve"
    "we're"      "who've"     "your"
```

List of Japanese Stop Words

Get a list of Japanese stop words using the `stopWords` function. For readability, reshape the output.

```
words = stopWords('Language','ja');
reshape([words strings(1,8)],[35 11])
```

```
ans = 35x11 string
  Columns 1 through 7

    "あそこ"      "さらい"      "なかば"      "下"    "今"    "地"      "列"
    "あたり"      "さん"       "なに"       "字"    "部"    "員"      "事"
    "あちら"      "しかた"      "など"       "年"    "課"    "線"      "士"
    "あっち"      "しよう"      "なん"       "月"    "係"    "点"      "台"
    "あと"       "すか"       "はじめ"      "日"    "外"    "書"      "集"
    "あな"       "ずつ"       "はず"       "時"    "類"    "品"      "様"
    "あなた"      "すね"       "はるか"      "分"    "達"    "力"      "所"
    "あれ"       "すべて"      "ひと"       "秒"    "気"    "法"      "歴"
    "いくつ"      "ぜんぶ"      "ひとつ"      "週"    "室"    "感"      "器"
    "いつ"        "そう"        "ふく"        "火"     "口"     "作"      "名"
    "いま"        "そこ"        "ぶり"        "水"     "誰"     "元"      "情"
    "いや"        "そちら"       "べつ"        "木"     "用"     "手"      "連"
    "いろいろ"     "そっち"       "へん"        "金"     "界"     "数"      "毎"
    "うち"        "そで"        "ぺん"        "土"     "会"     "彼"      "式"
    "おおまか"     "それ"        "ほう"        "国"     "首"     "彼女"     "簿"
    "おまえ"       "それぞれ"     "ほか"        "都"     "男"     "子"      "回"
    "おれ"        "それなり"     "まさ"        "道"     "女"     "内"      "匹"
    "がい"        "たくさん"     "まし"        "府"     "別"     "楽"      "個"
    "かく"        "たち"        "まとも"       "県"     "話"     "喜"      "席"
    "かたち"       "たび"        "まま"        "市"     "私"     "怒"      "束"
    "かやの"       "ため"        "みたい"       "区"     "屋"     "哀"      "歳"
    "から"        "だめ"        "みつ"        "町"     "店"     "輪"      "目"
    "がら"        "ちゃ"        "みなさん"     "村"     "家"     "頃"      "通"
    "きた"        "ちゃん"       "みんな"       "各"     "場"     "化"      "面"
    "くせ"        "てん"        "もと"        "第"     "等"     "境"      "円"
    "ここ"        "とおり"       "もの"        "方"     "見"     "俺"      "玉"
    "こっち"       "とき"        "もん"        "何"     "際"     "奴"      "枚"
    "こと"        "どこ"        "やつ"        "的"     "観"     "高"      "前"
    "ごと"        "どこか"       "よう"        "度"     "段"     "校"      "後"
    "こちら"       "ところ"       "よそ"        "文"     "略"     "婦"      "左"

  Columns 8 through 11

    "秋"       "本当"      "う"        "どう"
    "冬"       "確か"      "え"        "な"
    "一"       "時点"      "お"        "ない"
    "二"       "全部"      "か"        "なり"
    "三"       "関係"      "が"        "なる"
    "四"       "近く"      "こそ"       "に"
    "五"       "方法"      "この"       "ね"
    "六"       "我々"      "さ"        "の"
    "七"       "違い"      "さえ"       "ので"
    "八"       "多く"      "し"        "のに"
    "九"       "扱い"      "しか"       "は"
    "十"       "新た"      "する"       "ばかり"
    "百"       "その後"     "ず"        "へ"
    "千"       "半ば"      "せる"       "ほど"
    "万"       "結局"      "そして"      "ます"
    "億"       "様々"      "その"       "ませ"
    "兆"       "以前"      "た"        "また"
    "下記"      "以後"      "たい"       "まで"
    "上記"      "以降"      "ただ"       "も"
    "時間"      "未満"      "だ"        "や"
    "今回"      "以上"      "だけ"       "やら"
    "前回"      "以下"      "だに"       "よ"
    "場合"      "幾つ"      "だの"       "より"
    "一つ"      "毎日"      "ち"        "れる"
    "年生"      "自体"      "って"       "わ"
    "自分"      "向こう"     "て"        "を"
    "ヶ所"      "何人"      "で"        "ん"
    "ヵ所"      "手段"      "でし"       ""
    "カ所"      "同じ"      "です"       ""
    "箇所"      "感じ"      "では"       ""
      ⋮
```

List of German Stop Words

Get a list of German stop words using the `stopWords` function. For readability, reshape the output.

```
words = stopWords('Language','de');
reshape([words strings(1,7)],[25 8])
```

```
ans = 25x8 string
  Columns 1 through 6

    "ab"         "dann"      "doch"       "hattet"     "jene"        "mein"
    "aber"       "das"       "du"         "her"        "jenem"       "meine"
    "alle"       "dass"      "durch"      "hin"        "jenen"       "meinem"
    "allem"      "daß"       "ein"        "hätte"      "jener"       "meinen"
    "allen"      "dein"      "eine"       "hättest"    "jenes"       "meiner"
    "aller"      "deine"     "einem"      "hättet"     "kann"        "meines"
    "alles"      "deinem"    "einen"      "ich"        "kannst"      "mich"
    "als"        "deiner"    "einer"      "ihm"        "kein"        "mir"
    "also"       "deines"    "eines"      "ihn"        "keine"       "mit"
    "am"         "dem"       "er"         "ihr"        "keinem"      "muss"
    "an"         "den"       "es"         "ihre"       "keinen"      "musst"
    "andere"     "denn"      "euch"       "ihrem"      "keiner"      "musste"
    "anderem"    "der"       "euer"       "ihren"      "keines"      "muß"
    "anderen"    "derer"     "eure"       "ihrer"      "können"      "müssen"
    "anderer"    "des"       "eurem"      "ihres"      "könnte"      "müssten"
    "anderes"    "dessen"    "euren"      "im"         "könnten"     "nach"
    "auch"       "dich"      "eures"      "in"         "könntest"    "nicht"
    "auf"        "die"       "für"        "ins"        "ließ"        "nichts"
    "aus"        "dies"      "ganz"       "ist"        "man"         "noch"
    "bei"        "diese"     "gar"        "ja"         "manche"      "nun"
    "bin"        "diesem"    "habe"       "jede"       "manchem"     "nur"
    "bis"        "diesen"    "haben"      "jedem"      "manchen"     "ob"
    "bist"       "dieser"    "hat"        "jeden"      "mancher"     "oder"
    "da"         "dieses"    "hatte"      "jeder"      "manches"     "seid"
    "damit"      "dir"       "hattest"    "jedes"      "mehr"        "sein"

  Columns 7 through 8

    "seine"      "welcher"
    "seinem"     "welches"
    "seinen"     "wenn"
    "seiner"     "wer"
    "seines"     "werde"
    "sich"       "werden"
    "sie"        "weshalb"
    "sind"       "wie"
    "so"         "wieder"
    "um"         "wieso"
    "und"        "wir"
    "uns"        "wirst"
    "unter"      "wo"
    "vom"        "während"
    "von"        "zu"
    "vor"        "zum"
    "war"        "zur"
    "waren"      "über"
    "warst"      ""
    "warum"      ""
    "was"        ""
    "weil"       ""
    "welche"     ""
    "welchem"    ""
    "welchen"    ""
```

Input Arguments

language — Stop word language
'en' (default) | 'ja' | 'de' | 'ko'

Stop word language, specified as one of the following:

- `'en'` – English
- `'ja'` – Japanese
- `'de'` – German
- `'ko'` – Korean

For more information about language support in Text Analytics Toolbox, see “Language Considerations”.

More About

Language Considerations

The `stopWords` and `removeStopWords` functions support English, Japanese, German, and Korean stop words only.

To remove stop words from other languages, use `removeWords` and specify your own stop words to remove.

See Also
`bagOfNgrams` | `bagOfWords` | `normalizeWords` | `removeLongWords` | `removeShortWords` | `removeStopWords` | `removeWords` | `tokenizedDocument`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Language Considerations”
“Japanese Language Support”
“German Language Support”

Introduced in R2017b

string

Convert scalar document to string vector

Syntax

```
words = string(document)
```

Description

`words = string(document)` converts a scalar `tokenizedDocument` to a string array of words.

Examples

Convert Document to String

Convert a scalar tokenized document to a string array of words.

```
document = tokenizedDocument("an example of a short sentence")
```

```
document = 
  tokenizedDocument:

   6 tokens: an example of a short sentence
```

```
words = string(document)
```

```
words = 1x6 string
    "an"    "example"    "of"    "a"    "short"    "sentence"
```

Input Arguments

`document` — Input document
scalar `tokenizedDocument`

Input document, specified as a scalar `tokenizedDocument` object.

Output Arguments

`words` — Output words
string vector

Output words, returned as a string vector.

See Also

`context` | `doc2cell` | `doclength` | `joinWords` | `tokenizedDocument`

Topics
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

string

Convert parsed HTML tree to string

Syntax

```
str = string(tree)
```

Description

`str = string(tree)` converts the `htmlTree` object `tree` to string.

Tip Use the `string` function to help inspect the underlying HTML code of `htmlTree` objects. To navigate elements of `htmlTree` objects, use the `findElement` function.

Examples

Convert Parsed HTML Code to String

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using the `webread` function.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
```

Parse the HTML code using the `htmlTree` function.

```
tree = htmlTree(code);
```

Find all the paragraphs in the HTML tree using the `findElement` function. The paragraphs are the nodes with element name "P".

```
subtrees = findElement(tree,"P");
```

Convert the subtrees to string using the `string` function.

```
str = string(subtrees)
```

```
str = 26×1 string
    "<P class="h1">↵  <A href="../index.html" class="coming_from_product">Documentation</A>↵  <A
    "<P>Text Analytics Toolbox™ provides algorithms and visualizations for preprocessing, analyz
    "<P>Text Analytics Toolbox includes tools for processing raw text from sources such as equipm
    "<P>Using machine learning techniques such as LSA, LDA, and word embeddings, you can find clu
    "<P class="category_desc">Learn the basics of Text Analytics Toolbox</P>"
    "<P class="category_desc">Import text data into MATLAB<SUP>®</SUP> and preprocess it for ana
    "<P class="category_desc">Develop predictive models using topic models and word embeddings</P
    "<P class="category_desc">Visualize text data and models using word clouds and text scatter p
    "<P class="category_desc">Information on language support in Text Analytics Toolbox</P>"
    "<P>You clicked a link that corresponds to this MATLAB command:</P>"
    "<P>Run the command by entering it in the MATLAB Command Window. Web browsers do not support
    "<P>Choose a web site to get translated content where available and see local events and offe
    "<P>You can also select a web site from the following list:</P>"
    "<P>Select the China site (in Chinese or English) for best site performance. Other MathWorks
    "<P class="text-center">↵  <A href="#" class="worldwide_link">Contact your local office</A>↵↵
    "<P class="ff_section_title">Explore Products</P>"
    "<P class="ff_section_title">Try or Buy</P>"
    "<P class="ff_section_title">Learn to Use</P>"
    "<P class="ff_section_title">Get Support</P>"
    "<P class="ff_section_title">About <SPAN translate="no">MathWorks</SPAN></P>"
    "<P class="h4 add_font_futura_medium add_margin_0">↵  <SPAN translate="no">MathWorks</SPAN>↵↵
    "<P>↵  <EM>Accelerating the pace of engineering and science</EM>↵</P>"
    "<P><SPAN translate="no">MathWorks</SPAN> is the leading developer of mathematical computing
    "<P>↵  <A href="/discovery.html?s_tid=all_disc_mw_ff">Discover...</A>↵</P>"
    "<P class="copyright" translate="no">© 1994-2021 The MathWorks, Inc.</P>"
    "<P>↵  <EM>Join the conversation</EM>↵</P>"
```

Input Arguments

`tree` — HTML tree
`htmlTree` array

HTML tree, specified as an `htmlTree` array.

Output Arguments

`str` — String
string

String, returned as a string array with the same size as `tree`.

Compatibility Considerations

`string` function for `htmlTree` objects uses two spaces for indentation
Behavior changed in R2021a

The output of the `string` function for `htmlTree` objects is automatically indented for readability. Starting in R2021a, the function indents HTML code using two whitespace characters. In previous releases, the function indents HTML code with four spaces.

This change affects code that parses the HTML string directly. To parse and navigate HTML code, use `htmlTree` objects.

`string` function for `htmlTree` objects returns attributes in different order
Behavior changed in R2021a

When creating an `htmlTree` object, the software automatically parses the HTML element attributes of the input HTML code. Starting in R2021a, the software uses an updated algorithm to parse the HTML element attributes. This change can result in the `string` function returning HTML code with the attributes in a different order.

See Also

`extractHTMLText` | `findElement` | `getAttribute` | `htmlTree` | `ismissing` | `readPDFFormData` | `tokenizedDocument`

Topics
"Parse HTML and Extract Text Content"
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018b

string

Package: `textanalytics.unicode`

Convert UTF-32 representation to string

Syntax

`str = string(str32)`

Description

`str = string(str32)` converts the UTF-32 representation `str32` to string.

Examples

Convert UTF-32 String Representation to String

Convert the string `"Hello! 😀"` to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! 😀";
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Convert `str32` to string using the `string` function.

```
str = string(str32)
```

```
str = 
"Hello! 😀"
```

Input Arguments

`str32` — UTF-32 string representation
UTF32 array

UTF-32 string representation, specified as a `UTF32` array.

Output Arguments

`str` — String
string

String, returned as a string array with the same size as `str32`.

See Also

`UTF32` | `characterCategories` | `hex` | `textanalytics.unicode.nfd` | `tokenizedDocument`

Topics

"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Language Considerations"

Introduced in R2021a

textscatter

2-D scatter plot of text

Syntax

```
ts = textscatter(x,y,str)
ts = textscatter(xy,str)
ts = textscatter(ax, ___ )
ts = textscatter( ___ ,Name,Value)
```

Description

`ts = textscatter(x,y,str)` creates a text scatter plot with elements of `str` at the locations specified by the vectors `x` and `y`, and returns the resulting `TextScatter` object.

`ts = textscatter(xy,str)` uses locations specified by the rows of `xy`. This syntax is equivalent to `textscatter(xy(:,1),xy(:,2),str)`.

`ts = textscatter(ax, ___ )` plots into axes `ax`. You can use any input arguments from previous syntaxes.

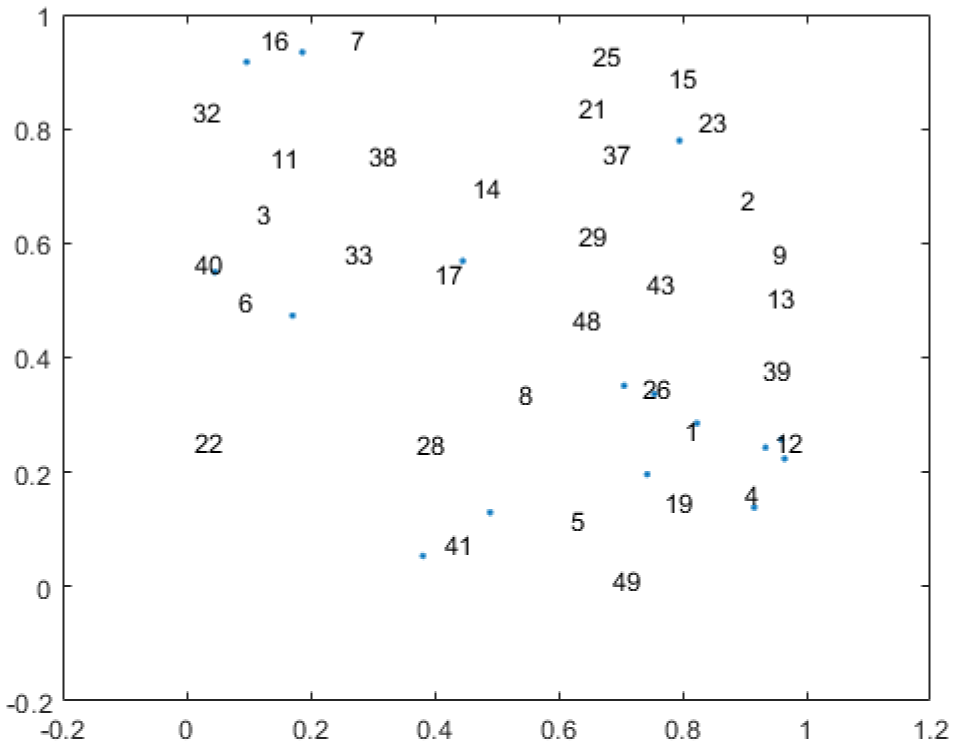
`ts = textscatter( ___ ,Name,Value)` specifies additional `TextScatter` properties using one or more name-value pair arguments.

Examples

Create Text Scatter Plot

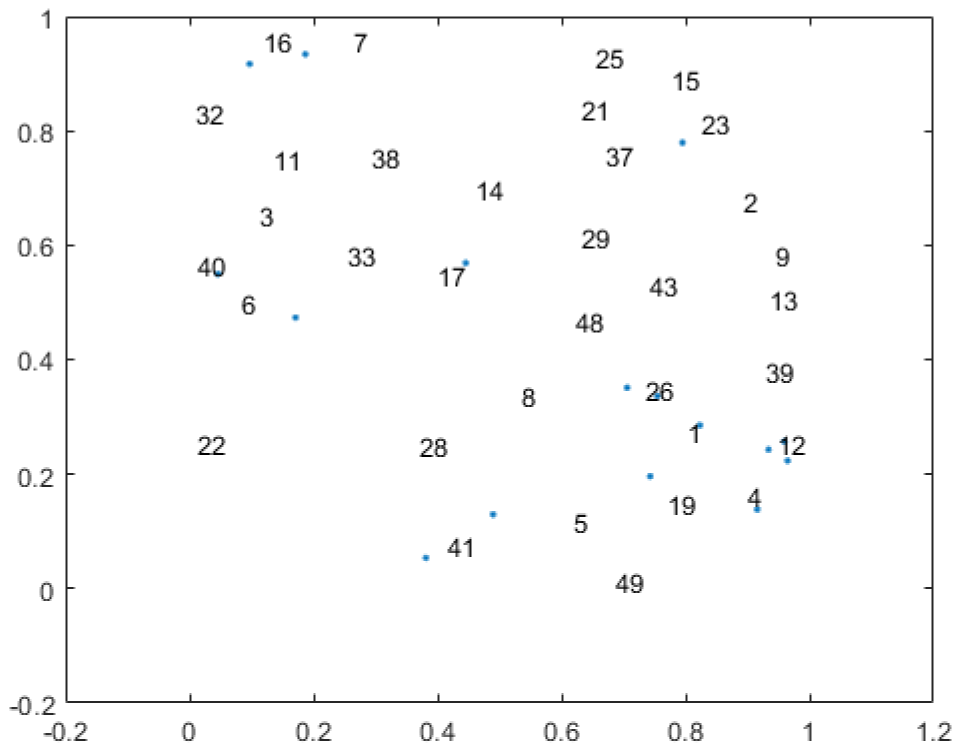
Plot a string array of numbers at random points on a text scatter plot.

```
x = rand(50,1);
y = rand(50,1);
str = string(1:50);
figure
textscatter(x,y,str);
```



Alternatively, you can pass the coordinates `x` and `y` as a matrix `xy`, where `x` and `y` are the columns of `xy`.

```
xy = [x y];
figure
textscatter(xy,str)
```



Specify Word Colors

Create text scatter plot of a word embedding and specify word colors.

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

Convert the first 500 words to vectors using `word2vec`. `V` is a matrix of word vectors of length 300.

```
words = emb.Vocabulary(1:500);
V = word2vec(emb,words);
size(V)
```

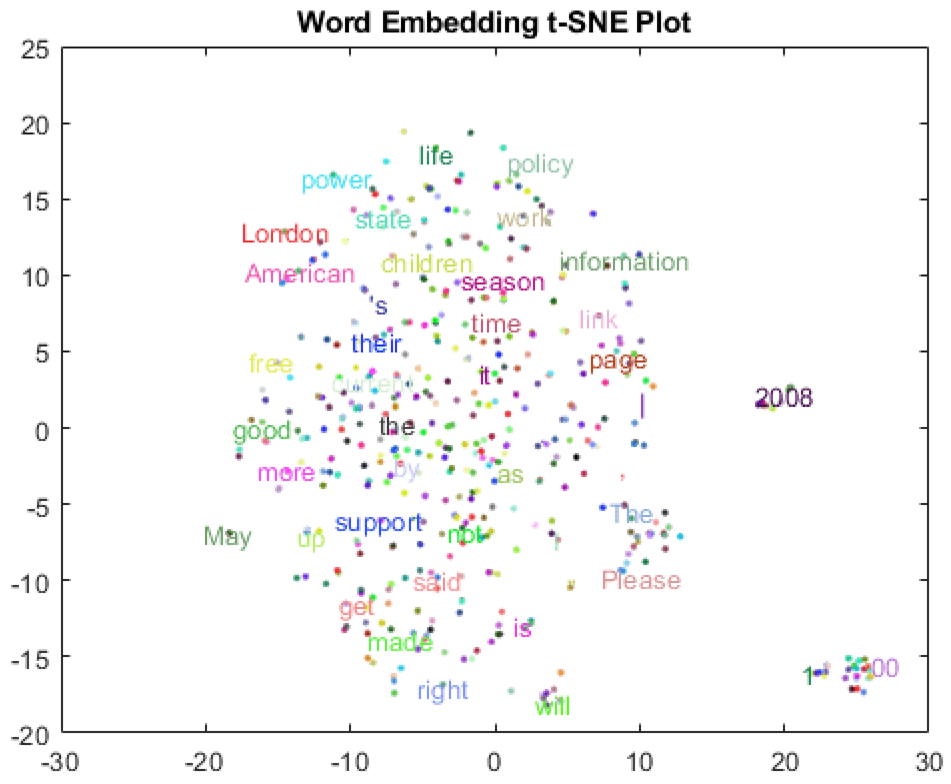
```
ans = 1×2

   500   300
```

Embed the word vectors in two-dimensional space using `tsne`.

```
XY = tsne(V);
```

Plot the words at the coordinates specified by XY in a 2-D text scatter plot. Specify the word colors to be random.

```
numWords = numel(words);
colorData = rand(numWords,3);
figure
textscatter(XY,words,'ColorData',colorData)
title("Word Embedding t-SNE Plot")
```



Input Arguments

x — *x* values

vector

x values, specified as a vector. `x`, `y`, and `str` must be of equal length.

Example: `[1 2 3]`

y — *y* values

vector

y values, specified as a vector. `x`, `y`, and `str` must be of equal length.

Example: `[1 2 3]`

xy — *x* and *y* values
matrix

x and *y* values, specified as a matrix with two columns. `xy(i,1)` and `xy(i,2)` correspond to the *x* and *y* values of the *i*th element of `str`, respectively. `xy` must have the `numel(str)` rows.

`textscatter(xy,str)` is equivalent to `textscatter(xy(:,1),xy(:,2),str)`.

Example: `[1 2 3]`

`str` — Input text
string vector | cell array of character vectors

Input text, specified as a string array or cell array of character vectors. `x`, `y`, and `str` must be of equal length.

Example: `["one" "two" "three"]`

Data Types: `string` | `cell`

ax — Axes object
axes object

Axes object. If you do not specify an axes object, then the function uses the current axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Marker','*'` specifies the markers to be asterisks.

The `TextScatter` object properties listed here are only a subset. For a complete list, see TextScatter Properties.

`TextDensityPercentage` — Percentage of text data to show
60 (default) | scalar from 0 through 100

Percentage of text data to show, specified as a scalar from 0 through 100. To show all text, set `TextDensityPercentage` to 100. To show no text, set `TextDensityPercentage` to 0.

If you set `TextDensityPercentage` to 100, then the software does not plot markers.

Example: 70

`MaxTextLength` — Maximum length of text labels
40 (default) | positive integer

Maximum length of text labels, specified as a positive integer. The software truncates the text labels to this length and adds ellipses at the point of truncation.

Example: 10

`MarkerColor` — Marker colors
`'auto'` (default) | `'none'` | RGB triplet

Marker colors, specified as one of these values:

- `'auto'` — For each marker, use the same color as the corresponding text labels.
- `'none'` — Do not show markers.
- RGB triplet — Use the same color for all the markers in the plot. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.

Example: `[1 0 0]`

`ColorData` — Text colors
`[ ]` (default) | RGB triplet | matrix of RGB triplets | categorical vector

Text colors, specified as one of these values:

- RGB triplet — Use the same color for all the text in the plot. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.
- Three-column matrix of RGB triplets — Use a different color for each text label in the plot. Each row of the matrix defines one color. The number of rows must equal the number of text labels.
- Categorical vector — Use a different color for each category in the vector. Specify `ColorData` as a vector the same length as `XData`. Specify the colors for each category using the `Colors` property

Example: `[1 0 0; 0 1 0; 0 0 1]`

`Colors` — Category colors
matrix of RGB triplets

Category colors, specified as a matrix of RGB triplets. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.

By default, `Colors` is equal to the `ColorOrder` property of the axes object.

Example: `[1 0 0; 0 1 0; 0 0 1]`

Output Arguments

`ts` — `TextScatter` object
`TextScatter` object

`TextScatter` object. Use `ts` to access and modify properties of the text scatter chart after it has been created. For more information, see TextScatter Properties.

See Also

`fastTextWordEmbedding` | `textscatter3` | `tokenizedDocument` | `word2vec` | `wordEmbedding` | `wordcloud`

Topics
“Visualize Text Data Using Word Clouds”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2017b

textscatter3

3-D scatter plot of text

Syntax

```
ts = textscatter3(x,y,z,str)
ts = textscatter3(xyz,str)
ts = textscatter3(ax, ___ )
ts = textscatter3( ___ ,Name,Value)
```

Description

`ts = textscatter3(x,y,z,str)` creates a 3-D text scatter plot with elements of `str` at the locations specified by the vectors `x`, `y`, and `z`.

`ts = textscatter3(xyz,str)` creates a 3-D text scatter plot with elements of `str` at the locations specified by the rows of `xyz`. This syntax is equivalent to `textscatter(xyz(:,1),xyz(:,2),xyz(:,3),str)`.

`ts = textscatter3(ax, ___ )` plots into axes object `ax`. Use this syntax with any of the input arguments in previous syntaxes.

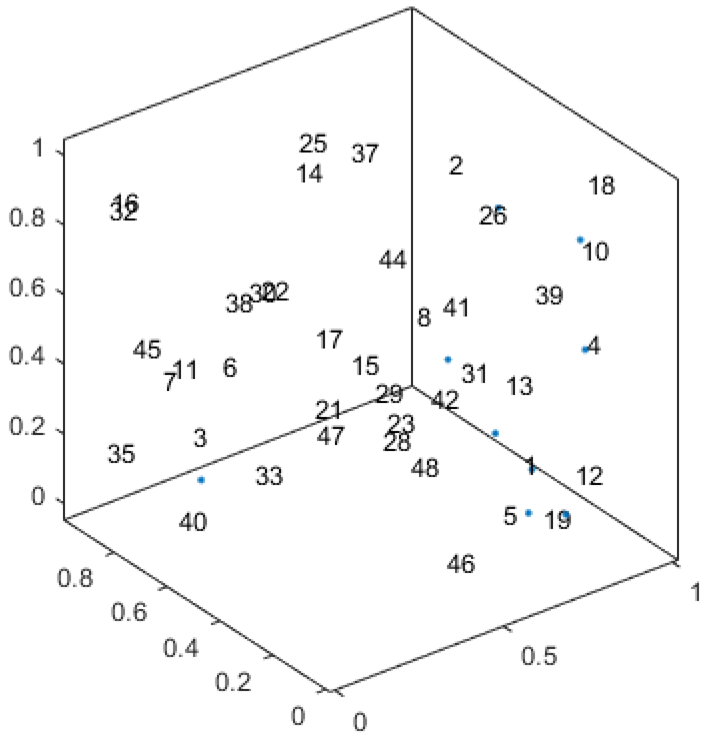
`ts = textscatter3( ___ ,Name,Value)` specifies additional `TextScatter` properties using one or more name-value pair arguments.

Examples

Create 3-D Text Scatter Plot

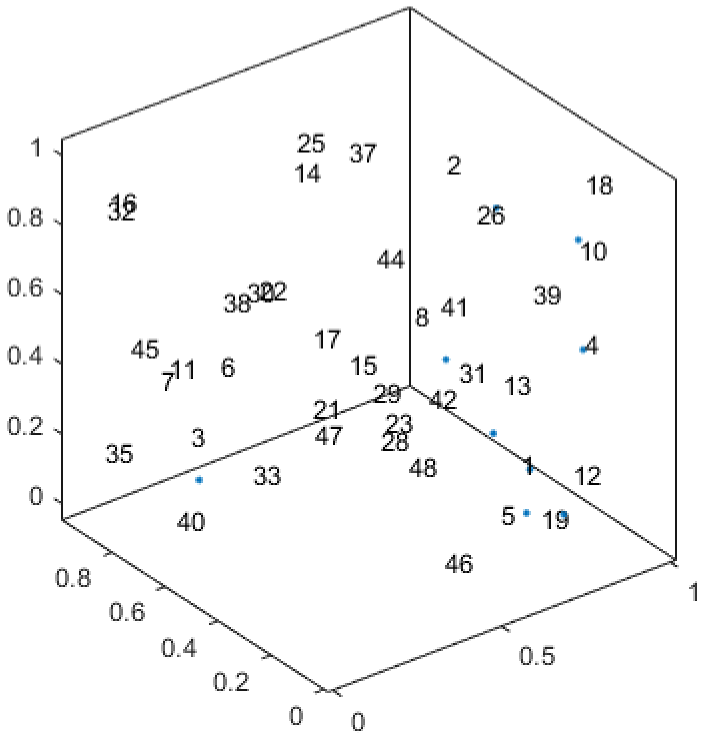
Plot a string array of numbers at random points on a 3-D text scatter plot.

```
x = rand(50,1);
y = rand(50,1);
z = rand(50,1);
str = string(1:50);
figure
textscatter3(x,y,z,str);
```



Alternatively, you can pass the coordinates `x`, `y`, and `z` as a matrix `xyz`, where `x`, `y`, and `z` are the columns of `xyz`.

```
xyz = [x y z];
figure
textscatter3(xyz,str)
```



Specify Word Colors

Create text scatter plot of a word embedding and specify word colors.

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding;
```

Convert the first 250 words to vectors using `word2vec`. `V` is a matrix of word vectors of length 300.

```
words = emb.Vocabulary(1:250);
V = word2vec(emb,words);
size(V)
```

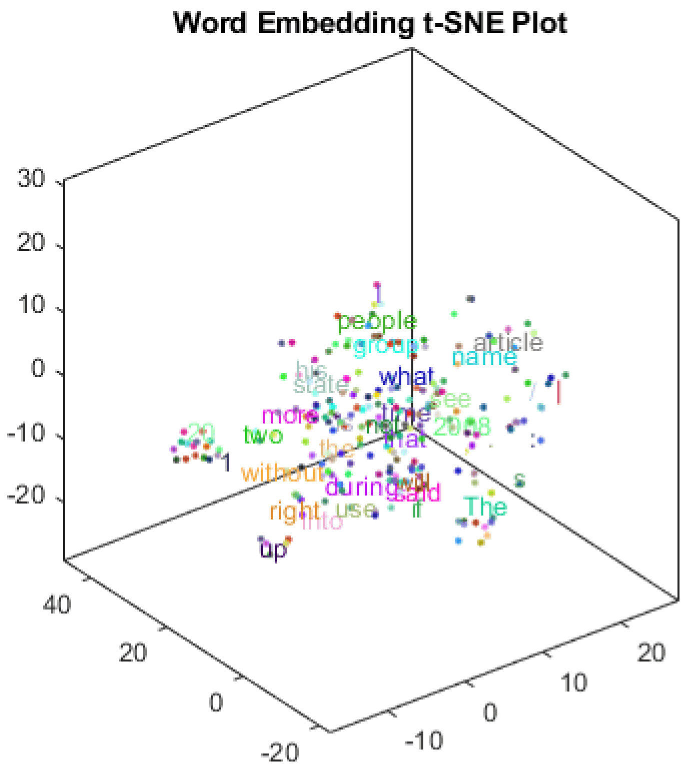
```
ans = 1×2

   250   300
```

Embed the word vectors in a 3-D space using `tsne`.

```
XYZ = tsne(V,'NumDimensions',3);
```

Plot the words at the coordinates specified by XYZ in a 3-D text scatter plot. Specify the word colors to be random.

```
numWords = numel(words);
colorData = rand(numWords,3);
figure
textscatter3(XYZ,words,'ColorData',colorData)
title("Word Embedding t-SNE Plot")
```



Input Arguments

x — *x* values
vector

x values, specified as a vector. x, y, z, and str must be of equal length.

Example: [1 2 3]

y — *y* values
vector

y values, specified as a vector. x, y, z, and str must be of equal length.

Example: [1 2 3]

z — *z* values
vector

z values, specified as a vector. `x`, `y`, `z`, and `str` must be of equal length.

Example: `[1 2 3]`

xyz — *x*, *y*, and *z* values
matrix

x, *y*, and *z* values, specified as a matrix. The first, second, and third columns of `xyz` correspond to the *x*, *y*, and *z* values, respectively.

`str` — Input text
string vector | cell array of character vectors

Input text, specified as a string array or cell array of character vectors. `x`, `y`, `z`, and `str` must be of equal length.

Example: `["one" "two" "three"]`

Data Types: `string` | `cell`

ax — Axes object
axes object

Axes object. If you do not specify an axes object, then the function uses the current axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Marker','*'` specifies the markers to be asterisks.

The `TextScatter` object properties listed here are only a subset. For a complete list, see TextScatter Properties.

`TextDensityPercentage` — Percentage of text data to show
60 (default) | scalar from 0 through 100

Percentage of text data to show, specified as a scalar from 0 through 100. To show all text, set `TextDensityPercentage` to 100. To show no text, set `TextDensityPercentage` to 0.

If you set `TextDensityPercentage` to 100, then the software does not plot markers.

Example: 70

`MaxTextLength` — Maximum length of text labels
40 (default) | positive integer

Maximum length of text labels, specified as a positive integer. The software truncates the text labels to this length and adds ellipses at the point of truncation.

Example: 10

`MarkerColor` — Marker colors
`'auto'` (default) | `'none'` | RGB triplet

Marker colors, specified as one of these values:

- `'auto'` — For each marker, use the same color as the corresponding text labels.
- `'none'` — Do not show markers.
- RGB triplet — Use the same color for all the markers in the plot. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.

Example: `[1 0 0]`

`ColorData` — Text colors
[ ] (default) | RGB triplet | matrix of RGB triplets | categorical vector

Text colors, specified as one of these values:

- RGB triplet — Use the same color for all the text in the plot. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.
- Three-column matrix of RGB triplets — Use a different color for each text label in the plot. Each row of the matrix defines one color. The number of rows must equal the number of text labels.
- Categorical vector — Use a different color for each category in the vector. Specify `ColorData` as a vector the same length as `XData`. Specify the colors for each category using the `Colors` property

Example: `[1 0 0; 0 1 0; 0 0 1]`

`Colors` — Category colors
matrix of RGB triplets

Category colors, specified as a matrix of RGB triplets. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.

By default, `Colors` is equal to the `ColorOrder` property of the axes object.

Example: `[1 0 0; 0 1 0; 0 0 1]`

Output Arguments

`ts` — `TextScatter` object
`TextScatter` object

`TextScatter` object. Use `ts` to access and modify properties of the text scatter chart after it has been created. For more information, see TextScatter Properties.

See Also
`fastTextWordEmbedding` | `textscatter` | `tokenizedDocument` | `word2vec` | `wordEmbedding` | `wordcloud`

Topics
“Visualize Text Data Using Word Clouds”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2017b

TextScatter Properties

Control text scatter chart appearance and behavior

Description

`TextScatter` properties control the appearance and behavior of `TextScatter` object. By changing property values, you can modify certain aspects of the text scatter chart.

Properties

Text

`TextData` — Text labels
string array | cell array of character vectors

Text labels, specified as a string array, or a cell array of character vectors.

Example: `["word1" "word2" "word3"]`

Data Types: `string` | `cell`

`TextDensityPercentage` — Percentage of text data to show
60 (default) | scalar from 0 through 100

Percentage of text data to show, specified as a scalar from 0 through 100. To show all text, set `TextDensityPercentage` to 100. To show no text, set `TextDensityPercentage` to 0.

If you set `TextDensityPercentage` to 100, then the software does not plot markers.

Example: 70

`MaxTextLength` — Maximum length of text labels
40 (default) | positive integer

Maximum length of text labels, specified as a positive integer. The software truncates the text labels to this length and adds ellipses at the point of truncation.

Example: 10

Font Style

`FontName` — Font name
system supported font name | `'FixedWidth'`

Font name, specified as the name of the font to use or `'FixedWidth'`. To display and print properly, the font name must be a font that your system supports. The default font depends on the specific operating system and locale.

To use a fixed-width font that looks good in any locale, use `'FixedWidth'`. The `'FixedWidth'` value relies on the root `FixedWidthFontName` property. Setting the root `FixedWidthFontName` property causes an immediate update of the display to use the new font.

Example: `'Cambria'`

FontSize — Font size
10 (default) | scalar value greater than zero

Font size, specified as a scalar value greater than zero in point units. One point equals 1/72 inch. To change the font units, use the FontUnits property.

Example: 12

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

FontAngle — Character slant
'normal' (default) | 'italic'

Character slant, specified as 'normal' or 'italic'. Not all fonts have both font styles. Therefore, the italic font might look the same as the normal font.

FontWeight — Thickness of text characters
'normal' (default) | 'bold'

Thickness of the text characters, specified as one of these values:

- 'normal' — Default weight as defined by the particular font
- 'bold' — Thicker character outlines than normal

MATLAB uses the FontWeight property to select a font from those available on your system. Not all fonts have a bold font weight. Therefore, specifying a bold font weight still can result in the normal font weight.

FontSmoothing — Smooth font character appearance
'on' (default) | 'off'

Smooth font character appearance, specified as one of these values:

- 'on' — Apply font smoothing. Reduce the appearance of jaggedness in the text characters to make the text easier to read.
- 'off' — Do not apply font smoothing.

Text Box

EdgeColor — Color of box outline
'none' (default) | RGB triplet | character vector of color name

Color of box outline, specified as 'none', a three-element RGB triplet, or a character vector of a color name. The default edge color of 'none' makes the box outline invisible.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	🟥
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	🟩
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	🟦
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	🟦
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	🟪
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	🟨
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	⬛
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	⬜

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	🟦
`[0.8500 0.3250 0.0980]`	`'#D95319'`	🟧
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	🟨
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	🟪
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	🟩
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	🟦
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	🟥

Example: `'blue'`

Example: `[0 0 1]`

`BackgroundColor` — Color of text box background
`'none'` (default) | `'data'` | RGB triplet

Color of text box background, specified as one of these values:

- `'none'` — Make the text box background transparent.
- `'data'` — Use background color specified by `ColorData`. The software automatically chooses a foreground to complement the background color.
- RGB triplet — Use the same color for all the markers in the plot. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.

Example: `[1 0 0]`

`Margin` — Space around text within text box
3 (default) | positive scalar

The space around the text within the text box, specified as a positive scalar in point units.

MATLAB uses the `Extent` property value plus the `Margin` property value to determine the size of the text box.

Example: 8

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Markers

`MarkerColor` — Marker colors

`'auto'` (default) | `'none'` | RGB triplet

Marker colors, specified as one of these values:

- `'auto'` — For each marker, use the same color as the corresponding text labels.
- `'none'` — Do not show markers.
- RGB triplet — Use the same color for all the markers in the plot. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.

Example: `[1 0 0]`

`MarkerSize` — Marker size

6 (default) | positive scalar

Marker size, specified as a positive scalar.

Example: 10

Data

`XData` — x values

`[]` (default) | scalar | vector

x values, specified as a scalar or a vector. The text scatter plot displays an individual marker for each value in `XData`.

The input argument `X` to the `textscatter` and `textscatter3` functions set the *x* values. `XData` and `YData` must have equal lengths.

Example: `[1 2 4 2 6]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `categorical` | `datetime` | `duration`

`XDataSource` — Variable linked to `XData`

`''` (default) | character vector containing MATLAB workspace variable name

Variable linked to `XData`, specified as a character vector containing a MATLAB workspace variable name. MATLAB evaluates the variable in the base workspace to generate the `XData`.

By default, there is no linked variable so the value is an empty character vector, `''`. If you link a variable, then MATLAB does not update the `XData` values immediately. To force an update of the data values, use the `refreshdata` function.

Note If you change one data source property to a variable that contains data of a different dimension, you might cause the function to generate a warning and not render the graph until you have changed all data source properties to appropriate values.

Example: `'x'`

`YData` — y values
`[]` (default) | scalar | vector

y values, specified as a scalar or a vector. The text scatter plot displays an individual marker for each value in `YData`.

The input argument `Y` to the `textscatter` and `textscatter3` functions set the *y* values. `XData` and `YData` must have equal lengths.

Example: `[1 3 3 4 6]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `categorical` | `datetime` | `duration`

`YDataSource` — Variable linked to `YData`
`''` (default) | character vector containing MATLAB workspace variable name

Variable linked to `YData`, specified as a character vector containing a MATLAB workspace variable name. MATLAB evaluates the variable in the base workspace to generate the `YData`.

By default, there is no linked variable so the value is an empty character vector, `''`. If you link a variable, then MATLAB does not update the `YData` values immediately. To force an update of the data values, use the `refreshdata` function.

Note If you change one data source property to a variable that contains data of a different dimension, you might cause the function to generate a warning and not render the graph until you have changed all data source properties to appropriate values.

Example: `'y'`

`ZData` — z values
`[]` (default) | scalar | vector

z values, specified as a scalar or a vector.

- For 2-D scatter plots, `ZData` is empty by default.
- For 3-D scatter plots, the input argument `Z` to the `scatter3` function sets the *z* values. `XData`, `YData`, and `ZData` must have equal lengths.

Example: `[1 2 2 1 0]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `categorical` | `datetime` | `duration`

`ZDataSource` — Variable linked to `ZData`
`''` (default) | character vector containing MATLAB workspace variable name

Variable linked to `ZData`, specified as a character vector containing a MATLAB workspace variable name. MATLAB evaluates the variable in the base workspace to generate the `ZData`.

By default, there is no linked variable so the value is an empty character vector, `''`. If you link a variable, then MATLAB does not update the `ZData` values immediately. To force an update of the data values, use the `refreshdata` function.

Note If you change one data source property to a variable that contains data of a different dimension, you might cause the function to generate a warning and not render the graph until you have changed all data source properties to appropriate values.

Example: `'z'`

`ColorData` — Text colors

`[]` (default) | RGB triplet | matrix of RGB triplets | categorical vector

Text colors, specified as one of these values:

- RGB triplet — Use the same color for all the text in the plot. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.
- Three-column matrix of RGB triplets — Use a different color for each text label in the plot. Each row of the matrix defines one color. The number of rows must equal the number of text labels.
- Categorical vector — Use a different color for each category in the vector. Specify `ColorData` as a vector the same length as `XData`. Specify the colors for each category using the `Colors` property

Example: `[1 0 0; 0 1 0; 0 0 1]`

`Colors` — Category colors

matrix of RGB triplets

Category colors, specified as a matrix of RGB triplets. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.5 0.6 0.7]`.

By default, `Colors` is equal to the `ColorOrder` property of the axes object.

Example: `[1 0 0; 0 1 0; 0 0 1]`

Visibility

`Visible` — State of visibility

`'on'` (default) | `'off'`

State of visibility, specified as one of these values:

- `'on'` — Display the object.
- `'off'` — Hide the object without deleting it. You still can access the properties of an invisible object.

Identifiers

`Type` — Type of graphics object

`'textscatter'`

This property is read-only.

Type of graphics object, returned as `'textscatter'`. Use this property to find all objects of a given type within a plotting hierarchy; for example, searching for the type using `findobj`.

Tag — User-specified tag

'' (default) | character vector

This property is read-only.

User-specified tag to associate with the object, specified as a character vector. Tags provide a way to identify graphics objects. Use this property to find all objects with a specific tag within a plotting hierarchy; for example, searching for the tag using `findobj`.

Example: `'January Data'`

`UserData` — Data to associate with object

`[]` (default) | any MATLAB data

This property is read-only.

Data to associate with the object, specified as any MATLAB data; for example, a scalar, vector, matrix, cell array, character array, table, or structure. MATLAB does not use this data.

To associate multiple sets of data or to attach a field name to the data, use the `getappdata` and `setappdata` functions.

Example: `1:100`

`DisplayName` — Text used for legend label

'' (default) | character vector

This property is read-only.

Text used for the legend label, specified as a character vector. If you do not specify the text, then the legend uses a label of the form `'dataN'`. The legend does not display until you call the `legend` command.

Example: `'Label Text'`

`Annotation` — Control for including or excluding object from legend

`Annotation` object

Control for including or excluding the object from a legend, returned as an `Annotation` object. Set the underlying `IconDisplayStyle` property to one of these values:

- `'on'` — Include the object in the legend (default).
- `'off'` — Do not include the object in the legend.

For example, exclude a stem chart from the legend.

```
p = plot(1:10,'DisplayName','Line Chart');
hold on
s = stem(1:10,'DisplayName','Stem Chart');
hold off
s.Annotation.LegendInformation.IconDisplayStyle = 'off';
legend('show')
```

Alternatively, you can control the items in a legend using the `legend` function. Specify the first input argument as a vector of the graphics objects to include.

```
p = plot(1:10,'DisplayName','Line Chart');
hold on
```

```
s = stem(1:10,'DisplayName','Stem Chart');
hold off
legend(p)
```

Parent/Child

`Parent` — Parent

`Axes` object | `PolarAxes` object | `Group` object | `Transform` object

Parent, specified as an `Axes`, `PolarAxes`, `Group`, or `Transform` object.

`Children` — Children

empty `GraphicsPlaceholder` array | `DataTip` object array

Children, returned as an empty `GraphicsPlaceholder` array or a `DataTip` object array. Use this property to view a list of data tips that are plotted on the chart.

You cannot add or remove children using the `Children` property. To add a child to this list, set the `Parent` property of the `DataTip` object to the chart object.

`HandleVisibility` — Visibility of object handle

`'on'` (default) | `'off'` | `'callback'`

Visibility of the object handle in the `Children` property of the parent, specified as one of these values:

- `'on'` — Object handle is always visible.
- `'off'` — Object handle is invisible at all times. This option is useful for preventing unintended changes to the UI by another function. Set the `HandleVisibility` to `'off'` to temporarily hide the handle during the execution of that function.
- `'callback'` — Object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line. This option blocks access to the object at the command-line, but allows callback functions to access it.

If the object is not listed in the `Children` property of the parent, then functions that obtain object handles by searching the object hierarchy or querying handle properties cannot return it. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

Hidden object handles are still valid. Set the root `ShowHiddenHandles` property to `'on'` to list all object handles regardless of their `HandleVisibility` property setting.

Interactive Control

`ButtonDownFcn` — Mouse-click callback

`''` (default) | function handle | cell array | character vector

Mouse-click callback, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- Character vector that is a valid MATLAB command or function, which is evaluated in the base workspace (not recommended)

Use this property to execute code when you click the object. If you specify this property using a function handle, then MATLAB passes two arguments to the callback function when executing the callback:

- Clicked object — You can access properties of the clicked object from within the callback function.
- Event data — This argument is empty for this property. Replace it with the tilde character (~) in the function definition to indicate that this argument is not used.

For more information on how to use function handles to define callback functions, see “Callback Definition”.

Note If the `PickableParts` property is set to `'none'` or if the `HitTest` property is set to `'off'`, then this callback does not execute.

Example: `@myCallback`

Example: `{@myCallback,arg3}`

`ContextMenu` — Context menu
empty `GraphicsPlaceholder` array (default) | `ContextMenu` object

Context menu, specified as a `ContextMenu` object. Use this property to display a context menu when you right-click the object. Create the context menu using the `uicontextmenu` function.

Note If the `PickableParts` property is set to `'none'` or if the `HitTest` property is set to `'off'`, then the context menu does not appear.

`Selected` — Selection state
`'off'` (default) | `'on'`

Selection state, specified as one of these values:

- `'on'` — Selected. If you click the object when in plot edit mode, then MATLAB sets its `Selected` property to `'on'`. If the `SelectionHighlight` property also is set to `'on'`, then MATLAB displays selection handles around the object.
- `'off'` — Not selected.

`SelectionHighlight` — Display of selection handles when selected
`'on'` (default) | `'off'`

Display of selection handles when selected, specified as one of these values:

- `'on'` — Display selection handles when the `Selected` property is set to `'on'`.
- `'off'` — Never display selection handles, even when the `Selected` property is set to `'on'`.

`DataTipTemplate` — Data tip content
`DataTipTemplate` object

Data tip content, specified as a `DataTipTemplate` object. You can control the content that appears in a data tip by modifying the properties of the underlying `DataTipTemplate` object. For a list of properties, see DataTipTemplate.

For an example of modifying data tips, see “Create Custom Data Tips”.

Note The `DataTipTemplate` object is not returned by `findobj` or `findall`, and it is not copied by `copyobj`.

Callback Execution Control

`PickableParts` — Ability to capture mouse clicks
`'visible'` (default) | `'none'`

Ability to capture mouse clicks, specified as one of these values:

- `'visible'` — Can capture mouse clicks when visible. The `Visible` property must be set to `'on'` and you must click a part of the `TextScatter` object that has a defined color. You cannot click a part that has an associated color property set to `'none'`. If the plot contains markers, then the entire marker is clickable if either the edge or the fill has a defined color. The `HitTest` property determines if the `TextScatter` object responds to the click or if an ancestor does.
- `'none'` — Cannot capture mouse clicks. Clicking the `TextScatter` object passes the click to the object below it in the current view of the figure window. The `HitTest` property of the `TextScatter` object has no effect.

`HitTest` — Response to captured mouse clicks
`'on'` (default) | `'off'`

Response to captured mouse clicks, specified as one of these values:

- `'on'` — Trigger the `ButtonDownFcn` callback of the `TextScatter` object. If you have defined the `UIContextMenu` property, then invoke the context menu.
- `'off'` — Trigger the callbacks for the nearest ancestor of the `TextScatter` object that has a `HitTest` property set to `'on'` and a `PickableParts` property value that enables the ancestor to capture mouse clicks.

Note The `PickableParts` property determines if the `TextScatter` object can capture mouse clicks. If it cannot, then the `HitTest` property has no effect.

`Interruptible` — Callback interruption
`'on'` (default) | `'off'`

Callback interruption, specified as `'on'` or `'off'`. The `Interruptible` property determines if a running callback can be interrupted.

Note There are two callback states to consider:

- The running callback is the currently executing callback.
- The interrupting callback is a callback that tries to interrupt the running callback.

Whenever MATLAB invokes a callback, that callback attempts to interrupt a running callback. The `Interruptible` property of the object owning the running callback determines if interruption is allowed. If interruption is not allowed, then the `BusyAction` property of the object owning the interrupting callback determines if it is discarded or put in the queue.

If the `ButtonDownFcn` callback of the `TextScatter` object is the running callback, then the `Interruptible` property determines if it another callback can interrupt it:

- `'on'` — Interruptible. Interruption occurs at the next point where MATLAB processes the queue, such as when there is a `drawnow`, `figure`, `getframe`, `waitfor`, or `pause` command.
  - If the running callback contains one of these commands, then MATLAB stops the execution of the callback at this point and executes the interrupting callback. MATLAB resumes executing the running callback when the interrupting callback completes. For more information, see “Interrupt Callback Execution”.
  - If the running callback does not contain one of these commands, then MATLAB finishes executing the callback without interruption.
- `'off'` — Not interruptible. MATLAB finishes executing the running callback without any interruptions.

`BusyAction` — Callback queuing

`'queue'` (default) | `'cancel'`

Callback queuing specified as `'queue'` or `'cancel'`. The `BusyAction` property determines how MATLAB handles the execution of interrupting callbacks.

Note There are two callback states to consider:

- The running callback is the currently executing callback.
- The interrupting callback is a callback that tries to interrupt the running callback.

Whenever MATLAB invokes a callback, that callback attempts to interrupt a running callback. The `Interruptible` property of the object owning the running callback determines if interruption is allowed. If interruption is not allowed, then the `BusyAction` property of the object owning the interrupting callback determines if it is discarded or put in the queue.

If the `ButtonDownFcn` callback of the `TextScatter` object tries to interrupt a running callback that cannot be interrupted, then the `BusyAction` property determines if it is discarded or put in the queue. Specify the `BusyAction` property as one of these values:

- `'queue'` — Put the interrupting callback in a queue to be processed after the running callback finishes execution. This is the default behavior.
- `'cancel'` — Discard the interrupting callback.

Creation and Deletion Control

`CreateFcn` — Creation callback

`''` (default) | function handle | cell array | character vector

Creation callback, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- Character vector that is a valid MATLAB command or function, which is evaluated in the base workspace (not recommended)

Use this property to execute code when you create the object. Setting the `CreateFcn` property on an existing object has no effect. You must define a default value for this property, or define this property using a `Name,Value` pair during object creation. MATLAB executes the callback after creating the object and setting all of its properties.

If you specify this callback using a function handle, then MATLAB passes two arguments to the callback function when executing the callback:

- Created object — You can access properties of the object from within the callback function. You also can access the object through the `CallbackObject` property of the root, which can be queried using the `gcbo` function.
- Event data — This argument is empty for this property. Replace it with the tilde character (~) in the function definition to indicate that this argument is not used.

For more information on how to use function handles to define callback functions, see “Callback Definition”.

Example: `@myCallback`

Example: `{@myCallback,arg3}`

`DeleteFcn` — Deletion callback
`''` (default) | function handle | cell array | character vector

Deletion callback, specified as one of these values:

- Function handle
- Cell array containing a function handle and additional arguments
- Character vector that is a valid MATLAB command or function, which is evaluated in the base workspace (not recommended)

Use this property to execute code when you delete the object MATLAB executes the callback before destroying the object so that the callback can access its property values.

If you specify this callback using a function handle, then MATLAB passes two arguments to the callback function when executing the callback:

- Deleted object — You can access properties of the object from within the callback function. You also can access the object through the `CallbackObject` property of the root, which can be queried using the `gcbo` function.
- Event data — This argument is empty for this property. Replace it with the tilde character (~) in the function definition to indicate that this argument is not used.

For more information on how to use function handles to define callback functions, see “Callback Definition”.

Example: `@myCallback`

Example: `{@myCallback,arg3}`

`BeingDeleted` — Deletion status
`'off'` (default) | `'on'`

Deletion status, returned as `'off'` or `'on'`. MATLAB sets the `BeingDeleted` property to `'on'` when the delete function of the object begins execution (see the `DeleteFcn` property). The `BeingDeleted` property remains set to `'on'` until the object no longer exists.

Check the value of the `BeingDeleted` property to verify that the object is not about to be deleted before querying or modifying it.

Compatibility Considerations

`UIContextMenu` property is not recommended
Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`bagOfNgrams` | `bagOfWords` | `textscatter` | `textscatter3` | `tokenizedDocument` | `wordCloudCounts` | `wordcloud`

Topics
"Visualize Text Data Using Word Clouds"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2017b

tfidf

Term Frequency–Inverse Document Frequency (tf-idf) matrix

Syntax

```
M = tfidf(bag)
M = tfidf(bag,documents)
M = tfidf( ___ ,Name,Value)
```

Description

`M = tfidf(bag)` returns a Term Frequency-Inverse Document Frequency (tf-idf) matrix based on the bag-of-words or bag-of-n-grams model `bag`.

`M = tfidf(bag,documents)` returns a tf-idf matrix for the documents in `documents` by using the inverse document frequency (IDF) factor computed from `bag`.

`M = tfidf( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Create Tf-idf Matrix

Create a Term Frequency–Inverse Document Frequency (tf-idf) matrix from a bag-of-words model.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Create a tf-idf matrix. View the first 10 rows and columns.

```
M = tfidf(bag);
full(M(1:10,1:10))
```

```
ans = 10×10

    3.6507    4.3438    2.7344    3.6507    4.3438    2.2644    3.2452    3.8918    2.4720    2.5
         0         0         0         0         0    4.5287         0         0         0
         0         0         0         0         0         0         0         0         0    2.5
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0         0    2.5
         0         0    2.7344         0         0         0         0         0         0
```

Create tf-idf Matrix from New Documents

Create a Term Frequency-Inverse Document Frequency (tf-idf) matrix from a bag-of-words model and an array of new documents.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model from the documents.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Create a tf-idf matrix for an array of new documents using the inverse document frequency (IDF) factor computed from `bag`.

```
newDocuments = tokenizedDocument([
    "what's in a name? a rose by any other name would smell as sweet."
    "if music be the food of love, play on."]);
M = tfidf(bag,newDocuments)
```

```
M = 
   (1,7)       3.2452
   (1,36)      1.2303
   (2,197)     3.4275
   (2,313)     3.6507
   (2,387)     0.6061
   (1,1205)    4.7958
   (1,1835)    3.6507
   (2,1917)    5.0370
```

Specify TF Weight Formulas

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Create a tf-idf matrix. View the first 10 rows and columns.

```
M = tfidf(bag);
full(M(1:10,1:10))
```

```
ans = 10×10

    3.6507    4.3438    2.7344    3.6507    4.3438    2.2644    3.2452    3.8918    2.4720    2.
         0         0         0         0         0    4.5287         0         0         0
         0         0         0         0         0         0         0         0         0    2.
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0         0    2.
         0         0    2.7344         0         0         0         0         0         0
```

You can change the contributions made by the TF and IDF factors to the tf-idf matrix by specifying the TF and IDF weight formulas.

To ignore how many times a word appears in a document, use the binary option of `'TFWeight'`. Create a tf-idf matrix and set `'TFWeight'` to `'binary'`. View the first 10 rows and columns.

```
M = tfidf(bag,'TFWeight','binary');
full(M(1:10,1:10))
```

```
ans = 10×10

    3.6507    4.3438    2.7344    3.6507    4.3438    2.2644    3.2452    1.9459    2.4720    2.5
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0         0         0         0         0    2.5
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0    2.2644         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0         0    2.5
         0         0    2.7344         0         0         0         0         0         0
```

Input Arguments

bag — Input bag-of-words or bag-of-n-grams model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object.

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Normalized',true` specifies to normalize the frequency counts.

`TFWeight` — Method to set term frequency factor
`'raw'` (default) | `'binary'` | `'log'`

Method to set term frequency (TF) factor, specified as the comma-separated pair consisting of `'TFWeight'` and one of the following:

- `'raw'` – Set the TF factor to the unchanged term counts.
- `'binary'` – Set the TF factor to the matrix of ones and zeros where the ones indicate whether a term is in a document.
- `'log'` – Set the TF factor to `1 + log(bag.Counts)`.

Example: `'TFWeight','binary'`

Data Types: `char`

`IDFWeight` — Method to compute inverse document frequency factor
`'normal'` (default) | `'textrank'` | `'classic-bm25'` | `'unary'` | `'smooth'` | `'max'` | `'probabilistic'`

Method to compute inverse document frequency factor, specified as the comma-separated pair consisting of `'IDFWeight'` and one of the following:

- `'textrank'` – Use TextRank IDF weighting [1]. For each term, set the IDF factor to
  - `log((N-NT+0.5)/(NT+0.5))` if the term occurs in more than half of the documents, where `N` is the number of documents in the input data and `NT` is the number of documents in the input data containing each term.
  - `IDFCorrection*avgIDF` if the term occurs in half of the documents or f, where `avgIDF` is the average IDF of all tokens.
- `'classic-bm25'` – For each term, set the IDF factor to `log((N-NT+0.5)/(NT+0.5))`.
- `'normal'` – For each term, set the IDF factor to `log(N/NT)`.
- `'unary'` – For each term, set the IDF factor to 1.
- `'smooth'` – For each term, set the IDF factor to `log(1+N/NT)`.
- `'max'` – For each term, set the IDF factor to `log(1+max(NT)/NT)`.
- `'probabilistic'` – For each term, set the IDF factor to `log((N-NT)/NT)`.

where `N` is the number of documents in the input data and `NT` is the number of documents in the input data containing each term.

Example: `'IDFWeight','smooth'`

Data Types: `char`

`IDFCorrection` — Inverse document frequency correction factor
0.25 (default) | nonnegative scalar

Inverse document frequency correction factor, specified as the comma-separated pair consisting of `'IDFCorrection'` and a nonnegative scalar.

This option only applies when `'IDFWeight'` is `'textrank'`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Normalized` — Option to normalize term counts
`false` (default) | `true`

Option to normalize term counts, specified as the comma-separated pair consisting of `'Normalized'` and `true` or `false`. If `true`, then the function normalizes each vector of term counts in the Euclidean norm.

Example: `'Normalized',true`

Data Types: `logical`

`DocumentsIn` — Orientation of output documents
`'rows'` (default) | `'columns'`

Orientation of output documents in the frequency count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Return a matrix of frequency counts with rows corresponding to documents.
- `'columns'` – Return a transposed matrix of frequency counts with columns corresponding to documents.

Data Types: `char`

`ForceCellOutput` — Indicator for forcing output to be returned as cell array
`false` (default) | `true`

Indicator for forcing output to be returned as cell array, specified as the comma separated pair consisting of `'ForceCellOutput'` and `true` or `false`.

Data Types: `logical`

Output Arguments

`M` — Output Term Frequency-Inverse Document Frequency matrix
sparse matrix | cell array of sparse matrices

Output Term Frequency-Inverse Document Frequency matrix, specified as a sparse matrix or a cell array of sparse matrices.

If `bag` is a non-scalar array or `'ForceCellOutput'` is `true`, then the function returns the outputs as a cell array of sparse matrices. Each element in the cell array is the tf-idf matrix calculated from the corresponding element of `bag`.

References

[1] Barrios, Federico, Federico López, Luis Argerich, and Rosa Wachenchauzer. "Variations of the Similarity Function of TextRank for Automated Summarization." *arXiv preprint arXiv:1602.03606* (2016).

See Also

`bagOfNgrams` | `bagOfWords` | `encode` | `tokenizedDocument` | `topkngrams` | `topkwords`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Analyze Text Data Using Topic Models”
“Analyze Text Data Using Multiword Phrases”
“Visualize Text Data Using Word Clouds”
“Classify Text Data Using Deep Learning”

Introduced in R2017b

tokenizedDocument

Array of tokenized documents for text analysis

Description

A tokenized document is a document represented as a collection of words (also known as tokens) which is used for text analysis.

Use tokenized documents to:

- Detect complex tokens in text, such as web addresses, emoticons, emoji, and hashtags.
- Remove words such as stop words using the `removeWords` or `removeStopWords` functions.
- Perform word-level preprocessing tasks such as stemming or lemmatization using the `normalizeWords` function.
- Analyze word and n-gram frequencies using `bagOfWords` and `bagOfNgrams` objects.
- Add sentence and part-of-speech details using the `addSentenceDetails` and `addPartOfSpeechDetails` functions.
- Add entity tags using the `addEntityDetails` function.
- View details about the tokens using the `tokenDetails` function.

The function supports English, Japanese, German, and Korean text. To learn how to use `tokenizedDocument` for other languages, see “Language Considerations” on page 1-404.

Creation

Syntax

```
documents = tokenizedDocument
documents = tokenizedDocument(str)
documents = tokenizedDocument(str,Name,Value)
```

Description

`documents = tokenizedDocument` creates a scalar tokenized document with no tokens.

`documents = tokenizedDocument(str)` tokenizes the elements of a string array and returns a tokenized document array.

`documents = tokenizedDocument(str,Name,Value)` specifies additional options using one or more name-value pair arguments.

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors | cell array of string arrays

Input text, specified as a string array, character vector, cell array of character vectors, or cell array of string arrays.

If the input text has not already been split into words, then `str` must be a string array, character vector, cell array of character vectors, or a cell array of string scalars.

Example: `["an example of a short document";"a second short document"]`

Example: `'an example of a single document'`

Example: `{'an example of a short document';'a second short document'}`

If the input text has already been split into words, then specify `'TokenizeMethod'` to be `'none'`. If `str` contains a single document, then it must be a string vector of words, a row cell array of character vectors, or a cell array containing a single string vector of words. If `str` contains multiple documents, then it must be a cell array of string arrays.

Example: `["an" "example" "document"]`

Example: `{'an','example','document'}`

Example: `{["an" "example" "of" "a" "short" "document"]}`

Example: `{["an" "example" "of" "a" "short" "document"];["a" "second" "short" "document"]}`

Data Types: `string` | `char` | `cell`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DetectPatterns',{'email-address','web-address'}` detects email addresses and web addresses

TokenizeMethod — Method to tokenize documents
`'unicode'` | `'mecab'` | `mecabOptions` object | `'none'`

Method to tokenize documents, specified as the comma-separated pair consisting of `'TokenizeMethod'` and one of the following:

- `'unicode'` – Tokenize input text using rules based on Unicode® Standard Annex #29 [1] and the ICU tokenizer [2]. If `str` is a cell array, then the elements of `str` must be string scalars or character vectors. If `'Language'` is `'en'` or`'de'`, then `'unicode'` is the default.
- `'mecab'` – Tokenize Japanese and Korean text using the MeCab tokenizer [3]. If `'Language'` is `'ja'` or `'ko'`, then `'mecab'` is the default.
- `mecabOptions` object – Tokenize Japanese and Korean text using the MeCab options specified by a `mecabOptions` object.
- `'none'` – Do not tokenize the input text.

If the input text has already been split into words, then specify `'TokenizeMethod'` to be `'none'`. If `str` contains a single document, then it must be a string vector of words, a row cell array of character vectors, or a cell array containing a single string vector of words. If `str` contains multiple documents, then it must be a cell array of string arrays.

`DetectPatterns` — Patterns of complex tokens to detect
`'all'` (default) | character vector | string array | cell array of character vectors

Patterns of complex tokens to detect, specified as the comma-separated pair consisting of `'DetectPatterns'` and `'none'`, `'all'`, or a string or cell array containing one or more of the following.

- `'email-address'` – Detect email addresses. For example, treat `"user@domain.com"` as a single token.
- `'web-address'` – Detect web addresses. For example, treat `"https://www.mathworks.com"` as a single token.
- `'hashtag'` – Detect hashtags. For example, treat `"#MATLAB"` as a single token.
- `'at-mention'` – Detect at-mentions. For example, treat `"@MathWorks"` as a single token.
- `'emoticon'` – Detect emoticons. For example, treat `":-D"` as a single token.

If `DetectPatterns` is `'none'`, then the function does not detect any complex token patterns. If `DetectPatterns` is `'all'`, then the function detects all the listed complex token patterns.

Example: `'DetectPatterns','hashtag'`

Example: `'DetectPatterns',{'email-address','web-address'}`

Data Types: `char` | `string` | `cell`

`CustomTokens` — Custom tokens to detect
`''` (default) | string array | character vector | cell array of character vectors | table

Custom tokens to detect, specified as the comma-separated pair consisting of `'CustomTokens'` and one of the following.

- A string array, character vector, or cell array of character vectors containing the custom tokens.
- A table containing the custom tokens in a column named `Token` and the corresponding token types a column named `Type`.

If you specify the custom tokens as a string array, character vector, or cell array of character vectors, then the function assigns token type `"custom"`. To specify a custom token type, use table input. To view the token types, use the `tokenDetails` function.

Example: `'CustomTokens',["C++" "C#"]`

Data Types: `char` | `string` | `table` | `cell`

`RegularExpressions` — Regular expressions to detect
`''` (default) | string array | character vector | cell array of character vectors | table

Regular expressions to detect, specified as the comma-separated pair consisting of `'RegularExpressions'` and one of the following.

- A string array, character vector, or cell array of character vectors containing regular expressions.
- A table containing regular expressions a column named `Pattern` and the corresponding token types in a column named `Type`.

If you specify the regular expressions as a string array, character vector, or cell array of character vectors, then the function assigns token type `"custom"`. To specify a custom token type, use table input. To view the token types, use the `tokenDetails` function.

Example: `'RegularExpressions',["ver:\d+" "rev:\d+"]`

Data Types: `char` | `string` | `table` | `cell`

`TopLevelDomains` — Top-level domains to use for web address detection
character vector | string array | cell array of character vectors

Top-level domains to use for web address detection, specified as the comma-separated pair consisting of `'TopLevelDomains'` and a character vector, string array, or cell array of character vectors. By default, the function uses the output of `topLevelDomains`.

This option only applies if `'DetectPatterns'` is `'all'` or contains `'web-address'`.

Example: `'TopLevelDomains',["com" "net" "org"]`

Data Types: `char` | `string` | `cell`

Language — Language
`'en'` | `'ja'` | `'de'` | `'ko'`

Language, specified as the comma-separated pair consisting of `'Language'` and one of the following.

- `'en'` – English. This option also sets the default value for `'TokenizeMethod'` to `'unicode'`.
- `'ja'` – Japanese. This option also sets the default value for `'TokenizeMethod'` to `'mecab'`.
- `'de'` – German. This option also sets the default value for `'TokenizeMethod'` to `'unicode'`.
- `'ko'` – Korean. This option also sets the default value for `'TokenizeMethod'` to `'mecab'`.

If you do not specify a value, then the function detects the language from the input text using the `corpusLanguage` function.

This option specifies the language details of the tokens. To view the language details of the tokens, use `tokenDetails`. These language details determine the behavior of the `removeStopWords`, `addPartOfSpeechDetails`, `normalizeWords`, `addSentenceDetails`, and `addEntityDetails` functions on the tokens.

For more information about language support in Text Analytics Toolbox, see “Language Considerations”.

Example: `'Language','ja'`

Properties

`Vocabulary` — Unique words in the documents
string array

Unique words in the documents, specified as a string array. The words do not appear in any particular order.

Data Types: `string`

Object Functions

Preprocessing

erasePunctuation	Erase punctuation from text and documents
removeStopWords	Remove stop words from documents
removeWords	Remove selected words from documents or bag-of-words model
normalizeWords	Stem or lemmatize words
correctSpelling	Correct spelling of words
replaceWords	Replace words in documents
replaceNgrams	Replace n-grams in documents
removeEmptyDocuments	Remove empty documents from tokenized document array, bag-of-words model, or bag-of-n-grams model
lower	Convert documents to lowercase
upper	Convert documents to uppercase

Tokens Details

tokenDetails	Details of tokens in tokenized document array
addSentenceDetails	Add sentence numbers to documents
addPartOfSpeechDetails	Add part-of-speech tags to documents
addLanguageDetails	Add language identifiers to documents
addTypeDetails	Add token type details to documents
addLemmaDetails	Add lemma forms of tokens to documents
addEntityDetails	Add entity tags to documents

Export

writeTextDocument	Write documents to text file

Manipulation and Conversion

doclength	Length of documents in document array
context	Search documents for word or n-gram occurrences in context
joinWords	Convert documents to string by joining words
doc2cell	Convert documents to cell array of string vectors
string	Convert scalar document to string vector
plus	Append documents
replace	Replace substrings in documents
docfun	Apply function to words in documents
regexprep	Replace text in words of documents using regular expression

Display

wordcloud	Create word cloud chart from text, bag-of-words model, bag-of-n-grams model, or LDA model

Examples

Tokenize Text

Create tokenized documents from a string array.

```
str = [
    "an example of a short sentence"
    "a second short sentence"]
```

```
str = 2x1 string
    "an example of a short sentence"
    "a second short sentence"
```

```
documents = tokenizedDocument(str)
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

Detect Complex Tokens

Create a tokenized document from the string `str`. By default, the function treats the hashtag `"#MATLAB"`, the emoticon `":-D"`, and the web address `"https://www.mathworks.com/help"` as single tokens.

```
str = "Learn how to analyze text in #MATLAB! :-D see https://www.mathworks.com/help/";
document = tokenizedDocument(str)
```

```
document = 
  tokenizedDocument:

   11 tokens: Learn how to analyze text in #MATLAB ! :-D see https://www.mathworks.com/help/
```

To detect only hashtags as complex tokens, specify the `'DetectPatterns'` option to be `'hashtag'` only. The function then tokenizes the emoticon `":-D"` and the web address `"https://www.mathworks.com/help"` into multiple tokens.

```
document = tokenizedDocument(str,'DetectPatterns','hashtag')
```

```
document = 
  tokenizedDocument:

   24 tokens: Learn how to analyze text in #MATLAB ! : - D see https : / / www . mathworks . com
```

Remove Stop Words from Documents

Remove the stop words from an array of documents using `removeStopWords`. The `tokenizedDocument` function detects that the documents are in English, so `removeStopWords` removes English stop words.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"]);
newDocuments = removeStopWords(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    3 tokens: example short sentence
    3 tokens: second short sentence
```

Stem Words in Documents

Stem the words in a document array using the Porter stemmer.

```
documents = tokenizedDocument([
    "a strongly worded collection of words"
    "another collection of words"]);
newDocuments = normalizeWords(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: a strongli word collect of word
    4 tokens: anoth collect of word
```

Specify Custom Tokens

The `tokenizedDocument` function, by default, splits words and tokens that contain symbols. For example, the function splits "C++" and "C#" into multiple tokens.

```
str = "I am experienced in MATLAB, C++, and C#.";
documents = tokenizedDocument(str)
```

```
documents = 
  tokenizedDocument:

   14 tokens: I am experienced in MATLAB , C + + , and C # .
```

To prevent the function from splitting tokens that contain symbols, specify custom tokens using the `'CustomTokens'` option.

```
documents = tokenizedDocument(str,'CustomTokens',["C++" "C#"])
```

```
documents = 
  tokenizedDocument:

   11 tokens: I am experienced in MATLAB , C++ , and C# .
```

The custom tokens have token type `"custom"`. View the token details. The column `Type` contains the token types.

```
tdetails = tokenDetails(documents)
```

```
tdetails=11×5 table
        Token         DocumentNumber    LineNumber       Type        Language
    _____________    ______________    __________    ___________    ________

    "I"                    1               1         letters           en
    "am"                   1               1         letters           en
    "experienced"          1               1         letters           en
    "in"                   1               1         letters           en
    "MATLAB"               1               1         letters           en
    ","                    1               1         punctuation       en
    "C++"                  1               1         custom            en
    ","                    1               1         punctuation       en
    "and"                  1               1         letters           en
    "C#"                   1               1         custom            en
    "."                    1               1         punctuation       en
```

To specify your own token types, input the custom tokens as a table with the tokens in a column named `Token`, and the types in a column named `Type`. To assign a custom type to a token that doesn't include symbols, include in the table too. For example, create a table that will assign "MATLAB", "C++", and "C#" to the `"programming-language"` token type.

```
T = table;
T.Token = ["MATLAB" "C++" "C#"]';
T.Type = ["programming-language" "programming-language" "programming-language"]'
```

```
T=3×2 table
     Token                Type
    ________    ______________________

    "MATLAB"    "programming-language"
    "C++"       "programming-language"
    "C#"        "programming-language"
```

Tokenize the text using the table of custom tokens and view the token details.

```
documents = tokenizedDocument(str,'CustomTokens',T);
tdetails = tokenDetails(documents)
```

```
tdetails=11×5 table
        Token         DocumentNumber    LineNumber            Type             Language
    _____________    ______________    __________    ____________________    ________

    "I"                    1               1         letters                    en
    "am"                   1               1         letters                    en
    "experienced"          1               1         letters                    en
    "in"                   1               1         letters                    en
    "MATLAB"               1               1         programming-language       en
    ","                    1               1         punctuation                en
    "C++"                  1               1         programming-language       en
    ","                    1               1         punctuation                en
    "and"                  1               1         letters                    en
    "C#"                   1               1         programming-language       en
    "."                    1               1         punctuation                en
```

Specify Custom Tokens Using Regular Expressions

The `tokenizedDocument` function, by default, splits words and tokens containing symbols. For example, the function splits the text `"ver:2"` into multiple tokens.

```
str = "Upgraded to ver:2 rev:3.";
documents = tokenizedDocument(str)
```

```
documents = 
  tokenizedDocument:

   9 tokens: Upgraded to ver : 2 rev : 3 .
```

To prevent the function from splitting tokens that have particular patterns, specify those patterns using the `'RegularExpressions'` option.

Specify regular expressions to detect tokens denoting version and revision numbers: strings of digits appearing after `"ver:"` and `"rev:"` respectively.

```
documents = tokenizedDocument(str,'RegularExpressions',["ver:\d+" "rev:\d+"])
```

```
documents = 
  tokenizedDocument:

   5 tokens: Upgraded to ver:2 rev:3 .
```

Custom tokens, by default, have token type `"custom"`. View the token details. The column `Type` contains the token types.

```
tdetails = tokenDetails(documents)
```

```
tdetails=5×5 table
      Token       DocumentNumber    LineNumber       Type        Language
    __________    ______________    __________    ___________    ________

    "Upgraded"          1               1         letters           en   
    "to"                1               1         letters           en   
    "ver:2"             1               1         custom            en   
    "rev:3"             1               1         custom            en   
    "."                 1               1         punctuation       en   
```

To specify your own token types, input the regular expressions as a table with the regular expressions in a column named `Pattern` and the token types in a column named `Type`.

```
T = table;
T.Pattern = ["ver:\d+" "rev:\d+"]';
T.Type = ["version" "revision"]'
```

```
T=2×2 table
     Pattern         Type    
    _________    __________

    "ver:\d+"    "version" 
    "rev:\d+"    "revision"
```

Tokenize the text using the table of custom tokens and view the token details.

```
documents = tokenizedDocument(str,'RegularExpressions',T);
tdetails = tokenDetails(documents)
```

```
tdetails=5×5 table
      Token        DocumentNumber    LineNumber       Type        Language
    __________    ______________    __________    ___________    ________

    "Upgraded"          1               1         letters           en   
    "to"                1               1         letters           en   
    "ver:2"             1               1         version           en   
    "rev:3"             1               1         revision          en   
    "."                 1               1         punctuation       en   
```

Search Documents for Word Occurrences

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Search for the word "life".

```
tbl = context(documents,"life");
head(tbl)
```

```
ans=8×3 table
                            Context                              Document    Word
    ________________________________________________________    ________    ____

    "consumst thy self single life ah thou issueless shalt "        9         10 
    "ainted counterfeit lines life life repair times pencil"       16         35 
    "d counterfeit lines life life repair times pencil pupi"       16         36 
    " heaven knows tomb hides life shows half parts write b"       17         14 
    "he eyes long lives gives life thee                    "       18         69 
    "tender embassy love thee life made four two alone sink"       45         23 
    "ves beauty though lovers life beauty shall black lines"       63         50 
    "s shorn away live second life second head ere beautys "       68         27 
```

View the occurrences in a string array.

```
tbl.Context
```

```
ans = 23x1 string
    "consumst thy self single life ah thou issueless shalt "
    "ainted counterfeit lines life life repair times pencil"
    "d counterfeit lines life life repair times pencil pupi"
    " heaven knows tomb hides life shows half parts write b"
    "he eyes long lives gives life thee                    "
    "tender embassy love thee life made four two alone sink"
    "ves beauty though lovers life beauty shall black lines"
    "s shorn away live second life second head ere beautys "
    "e rehearse let love even life decay lest wise world lo"
    "st bail shall carry away life hath line interest memor"
    "art thou hast lost dregs life prey worms body dead cow"
    "           thoughts food life sweetseasond showers gro"
    "tten name hence immortal life shall though once gone w"
    " beauty mute others give life bring tomb lives life fa"
    "ve life bring tomb lives life fair eyes poets praise d"
    " steal thyself away term life thou art assured mine li"
    "fe thou art assured mine life longer thy love stay dep"
    " fear worst wrongs least life hath end better state be"
    "anst vex inconstant mind life thy revolt doth lie o ha"
    " fame faster time wastes life thou preventst scythe cr"
    "ess harmful deeds better life provide public means pub"
    "ate hate away threw savd life saying                  "
    " many nymphs vowd chaste life keep came tripping maide"
```

Tokenize Japanese Text

Tokenize Japanese text using `tokenizedDocument`. The function automatically detects Japanese text.

```
str = [
    "恋に悩み、苦しむ。"
    "恋の悩みで苦しむ。"
    "空に星が輝き、瞬いている。"
    "空の星が輝きを増している。"];
documents = tokenizedDocument(str)
```

```
documents = 
  4x1 tokenizedDocument:

     6 tokens: 恋 に 悩み 、 苦しむ 。
     6 tokens: 恋 の 悩み で 苦しむ 。
    10 tokens: 空 に 星 が 輝き 、 瞬い て いる 。
    10 tokens: 空 の 星 が 輝き を 増し て いる 。
```

Tokenize German Text

Tokenize German text using `tokenizedDocument`. The function automatically detects German text.

```
str = [
    "Guten Morgen. Wie geht es dir?"
    "Heute wird ein guter Tag."];
documents = tokenizedDocument(str)
documents = 
  2x1 tokenizedDocument:

    8 tokens: Guten Morgen . Wie geht es dir ?
    6 tokens: Heute wird ein guter Tag .
```

More About

Language Considerations

The `tokenizedDocument` function has built-in rules for English, Japanese, German, and Korean only. For English and German text, the `'unicode'` tokenization method of `tokenizedDocument` detects tokens using rules based on Unicode Standard Annex #29 [1] and the ICU tokenizer [2], modified to better detect complex tokens such as hashtags and URLs. For Japanese and Korean text, the `'mecab'` tokenization method detects tokens using rules based on the MeCab tokenizer [3].

For other languages, you can still try using `tokenizedDocument`. If `tokenizedDocument` does not produce useful results, then try tokenizing the text manually. To create a `tokenizedDocument` array from manually tokenized text, set the `'TokenizeMethod'` option to `'none'`.

For more information, see “Language Considerations”.

Compatibility Considerations

`tokenizedDocument` detects Korean language
Behavior changed in R2019b

Starting in R2019b, `tokenizedDocument` detects the Korean language and sets the `'Language'` option to `'ko'`. This changes the default behavior of the `addSentenceDetails`, `addPartOfSpeechDetails`, `removeStopWords`, and `normalizeWords` functions for Korean document input. This change allows the software to use Korean-specific rules and word lists for analysis. If `tokenizedDocument` incorrectly detects text as Korean, then you can specify the language manually by setting the `'Language'` name-value pair of `tokenizedDocument`.

In previous versions, `tokenizedDocument` usually detects Korean text as English and sets the `'Language'` option to `'en'`. To reproduce this behavior, manually set the `'Language'` name-value pair of `tokenizedDocument` to `'en'`.

`tokenizedDocument` detects emoticons
Behavior changed in R2018b

Starting in R2018b, `tokenizedDocument`, by default, detects emoticon tokens. This behavior makes it easier to analyze text containing emoticons.

In R2017b and R2018a, `tokenizedDocument` splits emoticon tokens into multiple tokens. To reproduce this behavior, in `tokenizedDocument`, specify the `'DetectPatterns'` option to be `{'email-address','web-address','hashtag','at-mention'}`.

`tokenDetails` returns token type `emoji` for emoji characters
Behavior changed in R2018b

Starting in R2018b, `tokenizedDocument` detects emoji characters and the `tokenDetails` function reports these tokens with type `"emoji"`. This makes it easier to analyze text containing emoji characters.

In R2018a, `tokenDetails` reports emoji characters with type `"other"`. To find the indices of the tokens with type `"emoji"` or `"other"`, use the indices `idx = tdetails.Type == "emoji" | tdetails.Type == "other"`, where `tdetails` is a table of token details.

`tokenizedDocument` does not split at slash and colon characters between digits
Behavior changed in R2018b

Starting in R2018b, `tokenizedDocument` does not split at slash, backslash, or colon characters when they appear between two digits. This behavior produces better results when tokenizing text containing dates and times.

In previous versions, `tokenizedDocument` splits at these characters. To reproduce the behavior, tokenize the text manually or insert whitespace characters around slash, backslash, and colon characters before using `tokenizedDocument`.

References

[1] *Unicode Text Segmentation*. https://www.unicode.org/reports/tr29/

[2] *Boundary Analysis*. http://userguide.icu-project.org/boundaryanalysis

[3] *MeCab: Yet Another Part-of-Speech and Morphological Analyzer*. https://taku910.github.io/mecab/

See Also

`addEntityDetails` | `addPartOfSpeechDetails` | `addSentenceDetails` | `bagOfNgrams` | `bagOfWords` | `context` | `joinWords` | `normalizeWords` | `removeEmptyDocuments` | `removeStopWords` | `removeWords` | `replaceNgrams` | `replaceWords` | `tokenDetails`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Visualize Text Data Using Word Clouds”
“Analyze Text Data Using Topic Models”
“Analyze Text Data Using Multiword Phrases”
“Classify Text Data Using Deep Learning”
“Language Considerations”
“Japanese Language Support”
“German Language Support”

Introduced in R2017b

textanalytics.ja.mecabToLemma

Extract lemmata from MeCab output for Japanese

Syntax

`lemmata = textanalytics.ja.mecabToLemma(words,info)`

Description

`lemmata = textanalytics.ja.mecabToLemma(words,info)` extracts lemmata (normalized words) given MeCab output in the format returned by the MeCab-ipadic dictionary.

Input Arguments

`words` — Input tokens
string vector

Input tokens, specified as a string vector.

Data Types: `string`

`info` — Information struct
struct

Information struct with the following fields:

- `Feature` – String vector of tokens of the same size as `words` containing the MeCab output lines in ChaSen format without the split tokens themselves.
- `PartOfSpeech` – Numerical code used inside the MeCab-ipadic dictionary for the part-of-speech classification.

Data Types: `struct`

Output Arguments

`lemmata` — Extracted lemmata
string vector

Extracted lemmata, returned as a string vector the same size as `words`.

See Also

`addLemmaDetails` | `mecabOptions` | `normalizeWords` | `textanalytics.ja.mecabToNER` | `textanalytics.ja.mecabToPOS` | `tokenizedDocument`

Topics
"Japanese Language Support"
"Analyze Japanese Text Data"
"Language Considerations"
"Language-Independent Features"

Introduced in R2019b

textanalytics.ja.mecabToNER

Extract named entity information from MeCab output for Japanese

Syntax

`entities = textanalytics.ja.mecabToNER(words,info)`

Description

`entities = textanalytics.ja.mecabToNER(words,info)` extracts named entity information given MeCab output in the format returned by the MeCab-ipadic dictionary.

Input Arguments

`words` — Input tokens
string vector

Input tokens, specified as a string vector.

Data Types: `string`

`info` — Information struct
struct

Information struct with the following fields:

- `Feature` – String vector of tokens of the same size as `words` containing the MeCab output lines in ChaSen format without the split tokens themselves.
- `PartOfSpeech` – Numerical code used inside the MeCab-ipadic dictionary for the part-of-speech classification.

Data Types: `struct`

Output Arguments

`entities` — Extracted entity information
categorical vector

Extracted entity information, returned as a categorical vector the same size as `words`.

See Also
`addEntityDetails` | `mecabOptions` | `textanalytics.ja.mecabToLemma` | `textanalytics.ja.mecabToPOS` | `tokenizedDocument`

Topics
“Japanese Language Support”
“Analyze Japanese Text Data”
“Language Considerations”
“Language-Independent Features”

Introduced in R2019b

textanalytics.ja.mecabToPOS

Extract part-of-speech information from MeCab output for Japanese

Syntax

`posTags = textanalytics.ja.mecabToPOS(words,info)`

Description

`posTags = textanalytics.ja.mecabToPOS(words,info)` extracts part-of-speech information given MeCab output in the format returned by the MeCab-ipadic dictionary.

Input Arguments

`words` — Input tokens
string vector

Input tokens, specified as a string vector.

Data Types: `string`

`info` — Information struct
struct

Information struct with the following fields:

- `Feature` – String vector of tokens of the same size as `words` containing the MeCab output lines in ChaSen format without the split tokens themselves.
- `PartOfSpeech` – Numerical code used inside the MeCab-ipadic dictionary for the part-of-speech classification.

Data Types: `struct`

Output Arguments

`posTags` — Extracted part-of-speech information
categorical vector

Extracted part-of-speech information, returned as a categorical vector the same size as `words`.

See Also

`addPartOfSpeechDetails` | `mecabOptions` | `textanalytics.ja.mecabToLemma` | `textanalytics.ja.mecabToNER` | `tokenizedDocument`

Topics
“Japanese Language Support”
“Analyze Japanese Text Data”
“Language Considerations”
“Language-Independent Features”

Introduced in R2019b

UTF32

Unicode UTF-32 string representation

Description

The 32-bit Unicode transformation format (UTF-32) is a fixed length Unicode code point encoding that uses exactly 32 bits per code point.

Creation

Syntax

`str32 = textanalytics.unicode.UTF32(str)`

Description

`str32 = textanalytics.unicode.UTF32(str)` returns the Unicode UTF-32 representation of `str`. If `str` is an array, then `str32(i)` is the Unicode UTF-32 representation of the string `str(i)`.

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

Example: `["An example of a short sentence."; "A second short sentence."]`

Data Types: `string` | `char` | `cell`

Properties

`Data` — UTF-32 code points
`uint32` vector

UTF-32 code points, specified as a vector of integers with type `uint32`.

If the input string contains surrogate pairs, then the corresponding list of code points has a different length.

Data Types: `uint32`

Object Functions

characterCategories	Unicode character categories
hex	Convert UTF-32 representation to hexadecimal values
string	Convert UTF-32 representation to string

Examples

Convert Text to Unicode UTF-32 String Representation

Convert the string "`Hello! 😀`" to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! 😀";
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Get Unicode Character Categories

Convert the string "`Hello! 😀`" to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! 😀";
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Get the Unicode character categories of `str32` using the `characterCategories` function.

```
ucats = characterCategories(str32)
```

```
ucats = 1x1 cell array
    {[L    L    L    L    L    P    Z    S]}
```

The Unicode character categories "L", "P", "Z", and "S" correspond to "letter", "punctuation", "separator", and "symbol", respectively.

Get Detailed Unicode Character Categories

Convert the string "`Hello! 😀`" to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! 😀";
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:
    Data: [72 101 108 108 111 33 32 128512]
```

Get the Unicode character categories of `str32` using the `characterCategories` function. To return detailed Unicode character categories, set the `'Granularity'` option to `'detailed'`.

```
ucats = characterCategories(str32,'Granularity','detailed')
```

```
ucats = 1x1 cell array
    {[Lu    Ll    Ll    Ll    Ll    Po    Zs    So]}
```

The Unicode character categories "Lu", "Ll", "Po", "Zs", and "So" correspond to "uppercase letter", "lowercase letter", "other punctuation", "space separator", and "other symbol", respectively.

Convert UTF-32 String Representation to Hexadecimal Values

Convert the string `"Hello! □□` to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! □□;
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Convert `str32` to hexadecimal values using the `hex` function.

```
hexStr = hex(str32)
```

```
hexStr = 
" 0048  0065  006C  006C  006F  0021  0020 1F600"
```

Convert UTF-32 String Representation to String

Convert the string `"Hello! □□` to its Unicode UTF-32 string representation using the `textanalytics.unicode.UTF32` function.

```
str = "Hello! □□;
str32 = textanalytics.unicode.UTF32(str)
```

```
str32 = 
  UTF32 with properties:

    Data: [72 101 108 108 111 33 32 128512]
```

Convert `str32` to string using the `string` function.

```
str = string(str32)
str = 
"Hello! □□"
```

References

[1] *Unicode Standard Annex #19 UTF-32* https://www.unicode.org/reports/tr19/tr19-9.html

See Also

`characterCategories` | `hex` | `string` | `textanalytics.unicode.nfd` | `tokenizedDocument`

Topics
"Extract Text Data from Files"
"Prepare Text Data for Analysis"
"Language Considerations"

Introduced in R2021a

textrankKeywords

Extract keywords using TextRank

Syntax

```
tbl = textrankKeywords(documents)
tbl = textrankKeywords(documents,Name,Value)
```

Description

`tbl = textrankKeywords(documents)` extracts keywords and respective scores using TextRank. The function supports English, Japanese, German, and Korean text. For other languages, try using the `rakeKeywords` function instead.

`tbl = textrankKeywords(documents,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Extract Keywords Using TextRank

Create an array of tokenized document containing the text data.

```
textData = [
    "MATLAB provides really useful tools for engineers. Scientists use many useful tools in MATL
    "MATLAB and Simulink have many features. Use MATLAB and Simulink for engineering workflows."
    "Analyze text and images in MATLAB. Analyze text, images, and videos in MATLAB."];
documents = tokenizedDocument(textData);
```

Extract the keywords using the `textrankKeywords` function.

```
tbl = textrankKeywords(documents)
```

```
tbl=7×3 table
                Keyword                 DocumentNumber    Score 
    ________________________________    ______________    ______

    "many"      "useful"      "tools"         1           5.2174
    "useful"    "tools"       ""              1           3.8778
    "many"      "features"    ""              2           4.0815
    "text"      ""            ""              3                1
    "images"    ""            ""              3                1
    "MATLAB"    ""            ""              3                1
    "videos"    ""            ""              3                1
```

If a keyword contains multiple words, then the *i*th element of the string array corresponds to the *i*th word of the keyword. If the keyword has fewer words that the longest keyword, then remaining entries of the string array are the empty string `""`.

For readability, transform the multi-word keywords into a single sting using the `join` and `strip` functions.

```
if size(tbl.Keyword,2) > 1
    tbl.Keyword = strip(join(tbl.Keyword));
end
tbl
```

```
tbl=7×3 table
          Keyword           DocumentNumber    Score 
    ____________________    ______________    ______

    "many useful tools"           1           5.2174
    "useful tools"                1           3.8778
    "many features"               2           4.0815
    "text"                        3                1
    "images"                      3                1
    "MATLAB"                      3                1
    "videos"                      3                1
```

Specify Maximum Number of Keywords Per Document

Create an array of tokenized documents containing the text data.

```
textData = [
    "MATLAB provides really useful tools for engineers. Scientists use many useful MATLAB toolbox
    "MATLAB and Simulink have many features. Use MATLAB and Simulink for engineering workflows."
    "Analyze text and images in MATLAB. Analyze text, images, and videos in MATLAB."];
documents = tokenizedDocument(textData);
```

Extract the top two keywords using the `textrankKeywords` function and setting the `'MaxNumKeywords'` option to 2.

```
tbl = textrankKeywords(documents,'MaxNumKeywords',2)
```

```
tbl=5×3 table
                     Keyword                       DocumentNumber    Score 
    ______________________________________    ______________    ______

    "useful"    "MATLAB"      "toolboxes"           1           4.8695
    "useful"    ""            ""                    1           2.3612
    "many"      "features"    ""                    2           4.0815
    "text"      ""            ""                    3                1
    "images"    ""            ""                    3                1
```

If a keyword contains multiple words, then the *i*th element of the string array corresponds to the *i*th word of the keyword. If the keyword has fewer words that the longest keyword, then remaining entries of the string array are the empty string `""`.

For readability, transform the multi-word keywords into a single sting using the `join` and `strip` functions.

```
if size(tbl.Keyword,2) > 1
    tbl.Keyword = strip(join(tbl.Keyword));
```

```
end
tbl

tbl=5×3 table
             Keyword              DocumentNumber    Score 
    __________________________    ______________    ______

    "useful MATLAB toolboxes"           1           4.8695
    "useful"                            1           2.3612
    "many features"                     2           4.0815
    "text"                              3                1
    "images"                            3                1
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `textrankKeywords(documents,'MaxNumKeywords',20)` returns at most 20 keywords per document.

`MaxNumKeywords` — Maximum number of keywords to return per document
`Inf` (default) | positive integer

Maximum number of keywords to return per document, specified as the comma-separated pair consisting of `'MaxNumKeywords'` and a positive integer or `Inf`.

If `MaxNumKeywords` is `Inf`, then the function returns all identified keywords.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Window` — Size of co-occurrence window
2 (default) | positive integer | `Inf`

Size of co-occurrence window, specified as a the comma-separated pair consisting of `'Window'` and a positive integer or `Inf`.

When the window size is 2, the function considers a co-occurrence between two candidate keywords only when they appear consecutively in a document. When the window size is `Inf`, then the function considers a co-occurrence between two candidate keywords when they both appear in the same document.

Increasing the window size enables the function to find more co-occurrences between keywords which increases the keyword importance scores. This can result in finding more relevant keywords at the cost of potentially over-scoring less relevant keywords.

For more information, see "TextRank Keyword Extraction" on page 1-420.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`PartOfSpeech` — Part-of-speech tags
`["noun" "proper-noun" "adjective"]` (default) | string array | cell array of character vectors | character vector | categorical array

Part-of-speech tags to use to extract candidate keywords, specified as the comma-separated pair consisting of `'PartOfSpeech'` and a string array, cell array of character vectors, or a categorical array containing one or more of the following class names:

- `"adjective"` – Adjective
- `"adposition"` – Adposition
- `"adverb"` – Adverb
- `"auxiliary-verb"` – Auxiliary verb
- `"coord-conjunction"` – Coordinating conjunction
- `"determiner"` – Determiner
- `"interjection"` – Interjection
- `"noun"` – Noun
- `"numeral"` – Numeral
- `"particle"` – Particle
- `"pronoun"` – Pronoun
- `"proper-noun"` – Proper noun
- `"punctuation"` – Punctuation
- `"subord-conjunction"` – Subordinating conjucntion
- `"symbol"` – Symbol
- `"verb"` – Verb
- `"other"` – Other

If `PartOfSpeech` is a character vector, then it must correspond to a single part-of-speech tag.

For more information, see "TextRank Keyword Extraction" on page 1-420.

Data Types: `char` | `string` | `cell` | `categorical`

Output Arguments

`tbl` — Extracted keywords and scores
table

Extracted keywords and scores, returned as a table with the following variables:

- `Keyword` – Extracted keyword, specified as a 1-by-`maxNgramLength` string array, where `maxNgramLength` is the number of words in the longest keyword.

- `DocumentNumber` – Document number containing the corresponding keyword.
- `Score` – Score of keyword.

The function merges multiple keywords into a single keyword when they appear consecutively in the corresponding document.

If a keyword contains multiple words, then the *i*th element of the corresponding string array corresponds to the *i*th word of the keyword. If the keyword has fewer words that the longest keyword, then remaining entries of the string array are the empty string `""`.

For more information, see “TextRank Keyword Extraction” on page 1-420.

More About

Language Considerations

The `textrankKeywords` function supports English, Japanese, German, and Korean text only.

The `textrankKeywords` function extracts keywords by identifying candidate keywords based on their part-of-speech tag. The function uses part-of-speech tags given by the `addPartOfSpeechDetails` function which supports English, Japanese, German, and Korean text only.

For other languages, try using the `rakeKeywords` instead and specify an appropriate set of delimiters using the `'Delimiters'` and `'MergingDelimiters'` options.

Tips

- You can experiment with different keyword extraction algorithms to see what works best with your data. Because the TextRank keywords algorithm uses a part-of-speech tag-based approach to extract candidate keywords, the extracted keywords can be short. Alternatively, you can try extracting keywords using RAKE algorithm which extracts sequences of tokens appearing between delimiters as candidate keywords. To extract keywords using RAKE, use the `rakeKeywords` function. To learn more, see “Extract Keywords from Text Data Using RAKE”.

Algorithms

TextRank Keyword Extraction

For each document, the `textrankKeywords` function extracts keywords independently using the following steps based on [1]:

1. Determine candidate keywords:
   - Extract tokens with part-of-speech specified by the `'PartOfSpeech'` option.
2. Calculate scores for each candidate:
   - Create an undirected, unweighted graph with nodes corresponding to the candidate keywords.
   - Add edges between nodes where candidate keywords appear within a window of tokens, where the window size is given by the `'Window'` option.

   - Compute the centrality of each node using the PageRank algorithm and weight the scores according to the number of candidate keywords. For more information, see `centrality`.

3. Extract top keywords from candidates:

   - Select the top third of the candidate keywords according to their scores.
   - If any of the candidate keywords appear consecutively in a document, then merge them into a single keyword and sum the corresponding scores.
   - Return the top *k* keywords, where *k* is given by the `'MaxNumKeywords'` option.

Language Details

`tokenizedDocument` objects contain details about the tokens including language details. The language details of the input documents determine the behavior of `textrankKeywords`. The `tokenizedDocument` function, by default, automatically detects the language of the input text. To specify the language details manually, use the `'Language'` name-value pair argument of `tokenizedDocument`. To view the token details, use the `tokenDetails` function.

References

[1] Mihalcea, Rada, and Paul Tarau. "Textrank: Bringing order into text." In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pp. 404-411. 2004.

See Also

`extractSummary` | `rakeKeywords` | `textrankScores` | `tokenizedDocument`

Topics

“Extract Keywords from Text Data Using TextRank”
“Extract Keywords from Text Data Using RAKE”

Introduced in R2020b

textrankScores

Document scoring with TextRank algorithm

Syntax

```
scores = textrankScores(documents)
scores = textrankScores(bag)
```

Description

`scores = textrankScores(documents)` scores `documents` for importance according to pairwise similarity values using the TextRank algorithm. To compute similarities and importance scores, the function uses the BM25 and PageRank algorithms, respectively.

`scores = textrankScores(bag)` scores documents encoded by a bag-of-words or bag-of-n-grams model `bag`.

Examples

Importance of Documents

Create an array of tokenized documents.

```
str = [
    "the quick brown fox jumped over the lazy dog"
    "the fast brown fox jumped over the lazy dog"
    "the lazy dog sat there and did nothing"
    "the other animals sat there watching"];
documents = tokenizedDocument(str)
```

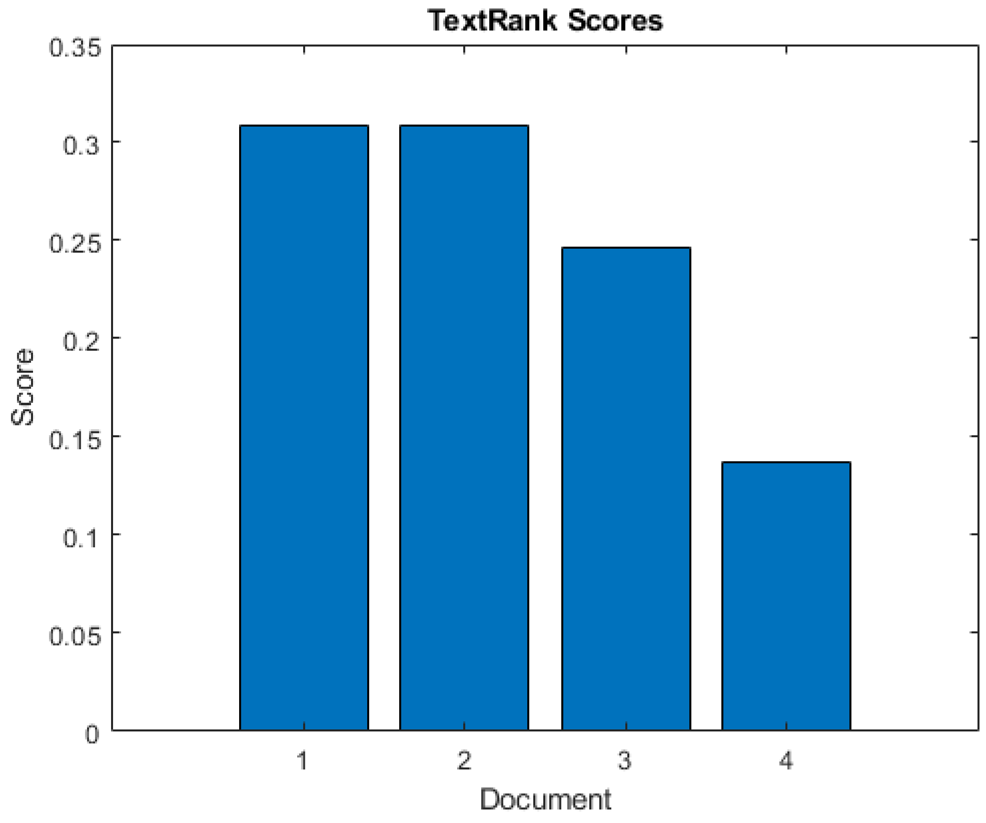
```
documents = 
  4x1 tokenizedDocument:

    9 tokens: the quick brown fox jumped over the lazy dog
    9 tokens: the fast brown fox jumped over the lazy dog
    8 tokens: the lazy dog sat there and did nothing
    6 tokens: the other animals sat there watching
```

Calculate the TextRank scores.

```
scores = textrankScores(documents);
```

Visualize the scores in a bar chart.

```
figure
bar(scores)
xlabel("Document")
ylabel("Score")
title("TextRank Scores")
```



Scores Using Bag-of-Words Model

Create a bag-of-words model from the text data in `sonnets.csv`.

```
filename = "sonnets.csv";
tbl = readtable(filename,'TextType','string');
textData = tbl.Sonnet;
documents = tokenizedDocument(textData);
bag = bagOfWords(documents)
```

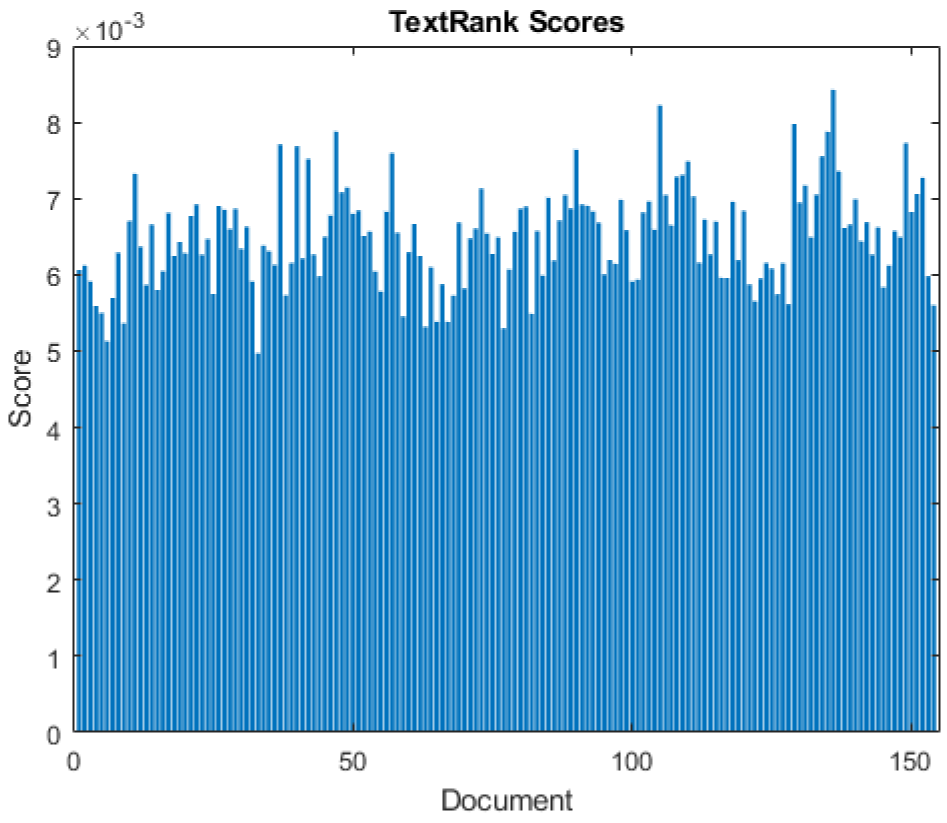
```
bag = 
  bagOfWords with properties:

          Counts: [154x3527 double]
      Vocabulary: [1x3527 string]
        NumWords: 3527
    NumDocuments: 154
```

Calculate the TextRank scores.

```
scores = textrankScores(bag);
```

Visualize the scores in a bar chart.

```
figure
bar(scores)
xlabel("Document")
ylabel("Score")
title("TextRank Scores")
```



Input Arguments

`documents` — Input documents

`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is not a `tokenizedDocument` array, then it must be a row vector representing a single document, where each element is a word. To specify multiple documents, use a `tokenizedDocument` array.

`bag` — Input model

`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

Output Arguments

`scores` — TextRank scores
vector

TextRank scores, returned as a *N*-by-1 vector, where `scores(i)` corresponds to the score for the `i`th input document and *N* is the number of input documents.

References

[1] Mihalcea, Rada, and Paul Tarau. "Textrank: Bringing order into text." In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pp. 404-411. 2004.

See Also

`bleuEvaluationScore` | `bm25Similarity` | `cosineSimilarity` | `extractSummary` | `lexrankScores` | `mmrScores` | `rougeEvaluationScore` | `tokenizedDocument`

Topics
“Sequence-to-Sequence Translation Using Attention”

Introduced in R2020a

tokenDetails

Details of tokens in tokenized document array

Syntax

```
tdetails = tokenDetails(documents)
```

Description

`tdetails = tokenDetails(documents)` returns a table of token details for the tokens in the `tokenizedDocument` array `documents`.

Examples

View Token Details of Documents

Create a tokenized document array.

```
str = [ ...
    "This is an example document. It has two sentences."
    "This document has one sentence and an emoticon. :)"
    "Here is another example document. :D"];
documents = tokenizedDocument(str);
```

View the token details of the first few tokens.

```
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×5 table
      Token        DocumentNumber    LineNumber       Type         Language
    __________    ______________    __________    ___________    ________

    "This"              1               1         letters           en
    "is"                1               1         letters           en
    "an"                1               1         letters           en
    "example"           1               1         letters           en
    "document"          1               1         letters           en
    "."                 1               1         punctuation       en
    "It"                1               1         letters           en
    "has"               1               1         letters           en
```

The `type` variable contains the type of each token. View the emoticons in the documents.

```
idx = tdetails.Type == "emoticon";
tdetails(idx,:)
```

```
ans=2×5 table
    Token    DocumentNumber    LineNumber      Type       Language
    _____    ______________    __________    ________    ________
    ":)"            2               1         emoticon       en
    ":D"            3               1         emoticon       en
```

Add Sentence Details to Documents

Create a tokenized document array.

```
str = [ ...
    "This is an example document. It has two sentences."
    "This document has one sentence."
    "Here is another example document. It also has two sentences."];
documents = tokenizedDocument(str);
```

Add sentence details to the documents using `addSentenceDetails`. This function adds the sentence numbers to the table returned by `tokenDetails`. View the updated token details of the first few tokens.

```
documents = addSentenceDetails(documents);
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×6 table
      Token        DocumentNumber    SentenceNumber    LineNumber       Type        Language
    __________    ______________    ______________    __________    ___________    ________

    "This"              1                 1               1         letters           en
    "is"                1                 1               1         letters           en
    "an"                1                 1               1         letters           en
    "example"           1                 1               1         letters           en
    "document"          1                 1               1         letters           en
    "."                 1                 1               1         punctuation       en
    "It"                1                 2               1         letters           en
    "has"               1                 2               1         letters           en
```

View the token details of the second sentence of the third document.

```
idx = tdetails.DocumentNumber == 3 & ...
    tdetails.SentenceNumber == 2;
tdetails(idx,:)
```

```
ans=6×6 table
       Token        DocumentNumber    SentenceNumber    LineNumber       Type        Language
    ___________    ______________    ______________    __________    ___________    ________

    "It"                 3                 2               1         letters           en
    "also"               3                 2               1         letters           en
    "has"                3                 2               1         letters           en
    "two"                3                 2               1         letters           en
    "sentences"          3                 2               1         letters           en
    "."                  3                 2               1         punctuation       en
```

Add Part-of-Speech Details to Documents

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

View the token details of the first few tokens.

```
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×5 table
       Token        DocumentNumber    LineNumber     Type      Language
    ___________    ______________    __________    _______    ________

    "fairest"            1               1         letters       en
    "creatures"          1               1         letters       en
    "desire"             1               1         letters       en
    "increase"           1               1         letters       en
    "thereby"            1               1         letters       en
    "beautys"            1               1         letters       en
    "rose"               1               1         letters       en
    "might"              1               1         letters       en
```

Add part-of-speech details to the documents using the `addPartOfSpeechDetails` function. This function first adds sentence information to the documents, and then adds the part-of-speech tags to the table returned by `tokenDetails`. View the updated token details of the first few tokens.

```
documents = addPartOfSpeechDetails(documents);
tdetails = tokenDetails(documents);
head(tdetails)
```

```
ans=8×7 table
       Token        DocumentNumber    SentenceNumber    LineNumber     Type      Language     Par
    ___________    ______________    ______________    __________    _______    ________    ___

    "fairest"            1                 1               1         letters       en       adje
    "creatures"          1                 1               1         letters       en       noun
    "desire"             1                 1               1         letters       en       noun
    "increase"           1                 1               1         letters       en       noun
    "thereby"            1                 1               1         letters       en       adver
    "beautys"            1                 1               1         letters       en       noun
    "rose"               1                 1               1         letters       en       noun
    "might"              1                 1               1         letters       en       auxi
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`tdetails` — Table of token details

table

Table of token details. `tdetails` has the following variables:

Name	Description
`Token`	Token text, returned as a string scalar.
`DocumentNumber`	Index of document that the token belongs to, returned as a positive integer.
`SentenceNumber`	Sentence number of token in document, returned as a positive integer. If these details are missing, then first add sentence details to `documents` using the `addSentenceDetails` function.
`LineNumber`	Line number of token in document, returned as a positive integer.
`Type`	The type of token, returned as one of the following: • `'letters'` – string of letter characters only • `'digits'` – string of digits only • `'punctuation'` – string of punctuation and symbol characters only • `'email-address'` – detected email address • `'web-address'` – detected web address • `'hashtag'` – detected hashtag (starts with `"#"` character followed by a letter) • `'at-mention'` – detected at-mention (starts with `"@"` character) • `'emoticon'` – detected emoticon • `'emoji'` – detected emoji • `'other'` – does not belong to the previous types and is not a custom type If these details are missing, then first add type details to `documents` using the `addTypeDetails` function.

Name	Description
`Language`	Language of the token, returned as one of the following: • `'en'` – English • `'ja'` – Japanese • `'de'` – German • `'ko'` – Korean These language details determine the behavior of the `removeStopWords`, `addPartOfSpeechDetails`, `normalizeWords`, `addSentenceDetails`, and `addEntityDetails` functions on the tokens. If these details are missing, then first add language details to `documents` using the `addLanguageDetails` function. For more information about language support in Text Analytics Toolbox, see “Language Considerations”.

Name	Description
`PartOfSpeech`	Part of speech tag, specified as a categorical from one of the following class names: • `"adjective"` – Adjective • `"adposition"` – Adposition • `"adverb"` – Adverb • `"auxiliary-verb"` – Auxiliary verb • `"coord-conjunction"` – Coordinating conjunction • `"determiner"` – Determiner • `"interjection"` – Interjection • `"noun"` – Noun • `"numeral"` – Numeral • `"particle"` – Particle • `"pronoun"` – Pronoun • `"proper-noun"` – Proper noun • `"punctuation"` – Punctuation • `"subord-conjunction"` – Subordinating conjucntion • `"symbol"` – Symbol • `"verb"` – Verb • `"other"` – Other If these details are missing, then first add part-of-speech details to `documents` using the `addPartOfSpeechDetails` function.
`Entity`	Entity tag, specified as one of the following: • `'location'` – detected location • `'organization'` – detected organization • `'person'` – detected person • `'other'` – detected entity, not belonging to the above categories • `'non-entity'` – no entity detected If these details are missing, then first add entity details to `documents` using the `addEntityDetails` function.
`Lemma`	Lemma form. If these details are missing, then first lemma details to `documents` using the `addLemmaDetails` function.

Compatibility Considerations

`tokenDetails` returns token type `emoji` for emoji characters
Behavior changed in R2018b

Starting in R2018b, `tokenizedDocument` detects emoji characters and the `tokenDetails` function reports these tokens with type `"emoji"`. This makes it easier to analyze text containing emoji characters.

In R2018a, `tokenDetails` reports emoji characters with type `"other"`. To find the indices of the tokens with type `"emoji"` or `"other"`, use the indices `idx = tdetails.Type == "emoji" | tdetails.Type == "other"`, where `tdetails` is a table of token details.

See Also

`addEntityDetails` | `addLanguageDetails` | `addLemmaDetails` | `addPartOfSpeechDetails` | `addSentenceDetails` | `addTypeDetails` | `normalizeWords` | `tokenizedDocument`

Topics
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”
“Language Considerations”
“Japanese Language Support”
“German Language Support”

Introduced in R2018a

topkwords

Most important words in bag-of-words model or LDA topic

Syntax

```
tbl = topkwords(bag)
tbl = topkwords(bag,k)

tbl = topkwords(ldaMdl,k,topicIdx)

tbl = topkwords( ___ ,Name,Value)
```

Description

`tbl = topkwords(bag)` returns a table of the five words with the largest word counts in bag-of-words model `bag`. The function, by default, is case sensitive.

`tbl = topkwords(bag,k)` returns a table of the `k` words with the largest word counts. The function, by default, is case sensitive.

`tbl = topkwords(ldaMdl,k,topicIdx)` returns a table of the `k` words with the highest probabilities in the latent Dirichlet allocation (LDA) topic `topicIdx` in the LDA model `ldaMdl`.

`tbl = topkwords( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Most Frequent Words of Bag-of-Words Model

Create a table of the most frequent words of a bag-of-words model.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
```

```
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Find the top five words.

```
T = topkwords(bag);
```

Find the top 20 words in the model.

```
k = 20;
T = topkwords(bag,k)
```

```
T=20×2 table
      Word       Count
    ________    _____

    "thy"        281 
    "thou"       234 
    "love"       162 
    "thee"       161 
    "doth"        88 
    "mine"        63 
    "shall"       59 
    "eyes"        56 
    "sweet"       55 
    "time"        53 
    "beauty"      52 
    "nor"         52 
    "art"         51 
    "yet"         51 
    "o"           50 
    "heart"       50 
      ⋮
```

Highest Probability Words of LDA Topic

Create a table of the words with highest probability of an LDA topic.

To reproduce the results, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents);
```

Fit an LDA model with 20 topics. To suppress verbose output, set `'Verbose'` to 0.

```
numTopics = 20;
mdl = fitlda(bag,numTopics,'Verbose',0);
```

Find the top 20 words of the first topic.

```
k = 20;
topicIdx = 1;
tbl = topkwords(mdl,k,topicIdx)
```

```
tbl=20×2 table
      Word        Score
    ________    _________

    "eyes"        0.11155
    "beauty"      0.05777
    "hath"       0.055778
    "still"      0.049801
    "true"       0.043825
    "mine"       0.033865
    "find"       0.031873
    "black"      0.025897
    "look"       0.023905
    "tis"        0.023905
    "kind"       0.021913
    "seen"       0.021913
    "found"      0.017929
    "sin"        0.015937
    "three"      0.013945
    "golden"    0.0099608
      ⋮
```

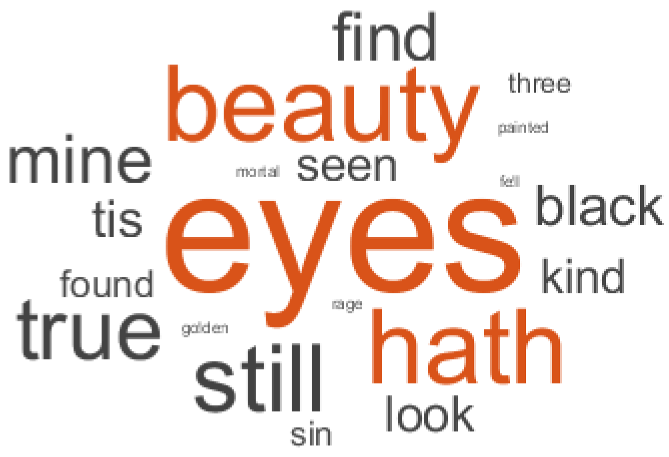
Find the top 20 words of the first topic and use inverse mean scaling on the scores.

```
tbl = topkwords(mdl,k,topicIdx,'Scaling','inversemean')
```

```
tbl=20×2 table
      Word       Score
    ________    ________

    "eyes"        1.2718
    "beauty"     0.59022
    "hath"        0.5692
    "still"      0.50269
    "true"       0.43719
    "mine"       0.32764
    "find"       0.32544
    "black"      0.25931
    "tis"        0.23755
    "look"       0.22519
    "kind"       0.21594
    "seen"       0.21594
    "found"      0.17326
    "sin"        0.15223
    "three"      0.13143
    "golden"    0.090698
        ⋮
```

Create a word cloud using the scaled scores as the size data.

```
figure
wordcloud(tbl.Word,tbl.Score);
```



Input Arguments

bag — Input bag-of-words model
`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` object.

k — Number of words
nonnegative integer

Number of words to return, specified as a positive integer.

Example: 20

`ldaMdl` — Input LDA model
`ldaModel` object

Input LDA model, specified as an `ldaModel` object.

`topicIdx` — Index of LDA topic
nonnegative integer

Index of LDA topic, specified as a nonnegative integer.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Scaling','inversemean'` specifies to use inverse mean scaling on the topic word probabilities.

Bag-of-Words Model Options

`IgnoreCase` — Option to ignore case
`false` (default) | `true`

Option to ignore case, specified as the comma-separated pair consisting of `'IgnoreCase'` and one of the following:

- `false` – treat words differing only by case as separate words.
- `true` – treat words differing only by case as the same word and merge counts.

This option supports bag-of-words input only.

`ForceCellOutput` — Indicator for forcing output to be returned as cell array
`false` (default) | `true`

Indicator for forcing output to be returned as cell array, specified as the comma separated pair consisting of `'ForceCellOutput'` and `true` or `false`.

This option supports bag-of-words input only.

Data Types: `logical`

LDA Model Options

`Scaling` — Scaling to apply to topic word probabilities
`'none'` (default) | `'inversemean'`

Scaling to apply to topic word probabilities, specified as the comma-separated pair consisting of `'Scaling'` and one of the following:

- `'none'` – Return posterior word probabilities.
- `'inversemean'` – Normalize the posterior word probabilities per topic by the geometric mean of the posterior probabilities for this word across all topics. The function uses the formula `Phi.*(log(Phi)-mean(log(Phi),1))`, where `Phi` corresponds to `ldaMdl.TopicWordProbabilities`.

This option supports LDA model input only.

Example: `'Scaling','inversemean'`

Data Types: `char`

Output Arguments

`tbl` — Table of top words

table | cell array of tables

Table of top words sorted in order of importance or a cell array of tables.

When the input is a bag-of-words model, the table has the following columns:

`Word`	Word specified as a string
`Count`	Number of times the word appears in the bag-of-words model

If `bag` is a non-scalar array or `'ForceCellOutput'` is `true`, then the function returns the outputs as a cell array of tables. Each element in the cell array is a table containing the top words of the corresponding element of `bag`.

When the input is an LDA model, the table has the following columns:

`Word`	Word specified as a string
`Score`	Word probability for the given LDA topic

Tips

- To find the most frequently seen n-grams in a bag-of-n-grams model, use `topkngrams`.

See Also

`bagOfNgrams` | `bagOfWords` | `ldaModel` | `removeInfrequentWords` | `removeWords` | `tfidf` | `tokenizedDocument` | `topkngrams`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2017b

topkngrams

Most frequent n-grams

Syntax

```
tbl = topkngrams(bag)
tbl = topkngrams(bag,k)
tbl = topkngrams(___,Name,Value)
```

Description

`tbl = topkngrams(bag)` returns a table listing the five most frequently seen n-grams in the bag-of-n-grams model `bag`. The function, by default, is case sensitive.

`tbl = topkngrams(bag,k)` lists the `k` most frequently seen n-grams in the bag-of-n-grams model `bag`. The function, by default, is case sensitive.

`tbl = topkngrams(___,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Most Frequent Bigrams of Bag-of-N-Grams Model

Create a table of the most frequent bigrams of a bag-of-n-grams model.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-n-grams model.

```
bag = bagOfNgrams(documents)
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154×8799 double]
      Vocabulary: [1×3092 string]
          Ngrams: [8799×2 string]
    NgramLengths: 2
       NumNgrams: 8799
    NumDocuments: 154
```

Find the top 5 bigrams.

```
tbl = topkngrams(bag)
```

```
tbl=5×3 table
         Ngram          Count    NgramLength
    ________________    _____    ___________

    "thou"    "art"      34           2     
    "mine"    "eye"      15           2     
    "thy"     "self"     14           2     
    "thou"    "dost"     13           2     
    "mine"    "own"      13           2     
```

Find the top 10 bigrams.

```
tbl = topkngrams(bag,10)
```

```
tbl=10×3 table
          Ngram          Count    NgramLength
    _________________    _____    ___________

    "thou"    "art"       34           2     
    "mine"    "eye"       15           2     
    "thy"     "self"      14           2     
    "thou"    "dost"      13           2     
    "mine"    "own"       13           2     
    "thy"     "sweet"     12           2     
    "thy"     "love"      11           2     
    "dost"    "thou"      10           2     
    "thou"    "wilt"      10           2     
    "love"    "thee"       9           2     
```

Count N-Grams of Different Lengths

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-n-grams model. To count n-grams of length 2 and 3 (bigrams and trigrams), specify `'NgramLengths'` to be the vector `[2 3]`.

```
bag = bagOfNgrams(documents,'NgramLengths',[2 3])
```

```
bag = 
  bagOfNgrams with properties:

          Counts: [154×18022 double]
```

```
      Vocabulary: [1×3092 string]
          Ngrams: [18022×3 string]
    NgramLengths: [2 3]
       NumNgrams: 18022
    NumDocuments: 154
```

View the 10 most common n-grams of length 2 (bigrams).

```
topkngrams(bag,10,'NGramLengths',2)
```

```
ans=10×3 table
             Ngram               Count    NgramLength
    _______________________    _____    ___________

    "thou"    "art"      ""     34            2     
    "mine"    "eye"      ""     15            2     
    "thy"     "self"     ""     14            2     
    "thou"    "dost"     ""     13            2     
    "mine"    "own"      ""     13            2     
    "thy"     "sweet"    ""     12            2     
    "thy"     "love"     ""     11            2     
    "dost"    "thou"     ""     10            2     
    "thou"    "wilt"     ""     10            2     
    "love"    "thee"     ""      9            2     
```

View the 10 most common n-grams of length 3 (trigrams).

```
 topkngrams(bag,10,'NGramLengths',3)
```

```
ans=10×3 table
               Ngram                  Count    NgramLength
    ____________________________    _____    ___________

    "thy"     "sweet"    "self"       4             3     
    "why"     "dost"     "thou"       4             3     
    "thy"     "self"     "thy"        3             3     
    "thou"    "thy"      "self"       3             3     
    "mine"    "eye"      "heart"      3             3     
    "thou"    "shalt"    "find"       3             3     
    "fair"    "kind"     "true"       3             3     
    "thou"    "art"      "fair"       2             3     
    "love"    "thy"      "self"       2             3     
    "thy"     "self"     "thou"       2             3     
```

Input Arguments

bag — Input bag-of-n-grams model
`bagOfNgrams` object

Input bag-of-n-grams model, specified as a `bagOfNgrams` object.

k — Number of n-grams
nonnegative integer

Number of n-grams to return, specified as a positive integer.

Example: 20

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NgramLengths',[2 3]` specifies to return the top bigrams and trigrams.

`NgramLengths` — N-gram lengths
positive integer | vector of positive integers

N-gram lengths, specified as the comma separated pair consisting of `'NgramLengths'` and a positive integer or a vector of positive integers.

If you specify `NgramLengths`, then the function returns n-grams of these lengths only. If you do not specify `NgramLengths`, then the function returns the top n-grams regardless of length.

Example: `[1 2 3]`

`IgnoreCase` — Option to ignore case
`false` (default) | `true`

Option to ignore case, specified as the comma-separated pair consisting of `'IgnoreCase'` and one of the following:

- `false` – treat n-grams differing only by case as separate n-grams.
- `true` – treat n-grams differing only by case as the same n-gram and merge counts.

`ForceCellOutput` — Indicator for forcing output to be returned as cell array
`false` (default) | `true`

Indicator for forcing output to be returned as cell array, specified as the comma separated pair consisting of `'ForceCellOutput'` and `true` or `false`.

Data Types: `logical`

Output Arguments

`tbl` — Table of top n-grams
table | cell array of tables

Table of top n-grams sorted in order of frequency or a cell array of tables.

The table has the following columns:

`Ngram`	N-gram specified as a string vector
`Count`	Number of times the n-gram appears in the bag-of-n-grams model.
`NgramLength`	Length of the n-gram.

If `bag` is a non-scalar array or `'ForceCellOutput'` is `true`, then the function returns the outputs as a cell array of tables. Each element in the cell array is a table containing the top n-grams of the corresponding element of `bag`.

See Also

`bagOfNgrams` | `bagOfWords` | `removeInfrequentNgrams` | `removeNgrams` | `tfidf` | `tokenizedDocument` | `topkwords`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"
"Analyze Text Data Using Topic Models"
"Analyze Text Data Using Multiword Phrases"
"Visualize Text Data Using Word Clouds"
"Classify Text Data Using Deep Learning"

Introduced in R2018a

topLevelDomains

List of top-level domains

Syntax

`domains = topLevelDomains`

Description

`domains = topLevelDomains` returns a string array of common top-level internet domain names which you can use to tokenize documents containing URLs.

Examples

List of Top-Level Domains

View list of top-level domains used to detect web addresses in strings. Reshape the output for readability.

```
domains = topLevelDomains;
reshape(domains, [], 5)
```

```
ans = 51x5 string
    "com"     "ck"    "hn"    "mp"    "si"
    "edu"     "cl"    "hr"    "mq"    "sj"
    "gov"     "cm"    "ht"    "mr"    "sk"
    "int"     "cn"    "hu"    "ms"    "sl"
    "mil"     "co"    "id"    "mt"    "sm"
    "net"     "cr"    "ie"    "mu"    "sn"
    "org"     "cu"    "il"    "mv"    "so"
    "info"    "cv"    "im"    "mw"    "sr"
    "ac"      "cw"    "in"    "mx"    "st"
    "ad"      "cx"    "io"    "my"    "su"
    "ae"      "cy"    "iq"    "mz"    "sv"
    "af"      "cz"    "ir"    "na"    "sx"
    "ag"      "de"    "is"    "nc"    "sy"
    "ai"      "dj"    "it"    "ne"    "sz"
    "am"      "dk"    "je"    "nf"    "tc"
    "ao"      "dm"    "jm"    "ng"    "td"
    "aq"      "do"    "jo"    "ni"    "tf"
    "ar"      "dz"    "jp"    "nl"    "tg"
    "as"      "ec"    "ke"    "no"    "th"
    "at"      "ee"    "kg"    "np"    "tj"
    "au"      "eg"    "kh"    "nr"    "tk"
    "aw"      "er"    "ki"    "nu"    "tl"
    "ax"      "es"    "km"    "nz"    "tm"
    "az"      "et"    "kp"    "om"    "tn"
    "ba"      "eu"    "kr"    "pa"    "to"
    "bb"      "fi"    "kw"    "pe"    "tr"
    "bd"      "fj"    "ky"    "pf"    "tt"
    "be"      "fk"    "kz"    "pg"    "tv"
    "bf"      "fm"    "la"    "ph"    "tw"
    "bg"      "fo"    "lb"    "pk"    "tz"
      ⋮
```

See Also

`addPartOfSpeechDetails` | `addSentenceDetails` | `addTypeDetails` | `tokenDetails` | `tokenizedDocument`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2018a

trainWordEmbedding

Train word embedding

Syntax

```
emb = trainWordEmbedding(filename)
emb = trainWordEmbedding(documents)
emb = trainWordEmbedding( ___ ,Name,Value)
```

Description

`emb = trainWordEmbedding(filename)` trains a word embedding using the training data stored in the text file `filename`. The file is a collection of documents stored in UTF-8 with one document per line and words separated by whitespace.

`emb = trainWordEmbedding(documents)` trains a word embedding using `documents` by creating a temporary file with `writeTextDocument`, and then trains an embedding using the temporary file.

`emb = trainWordEmbedding( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments. For example, `'Dimension',50` specifies the word embedding dimension to be 50.

Examples

Train Word Embedding from File

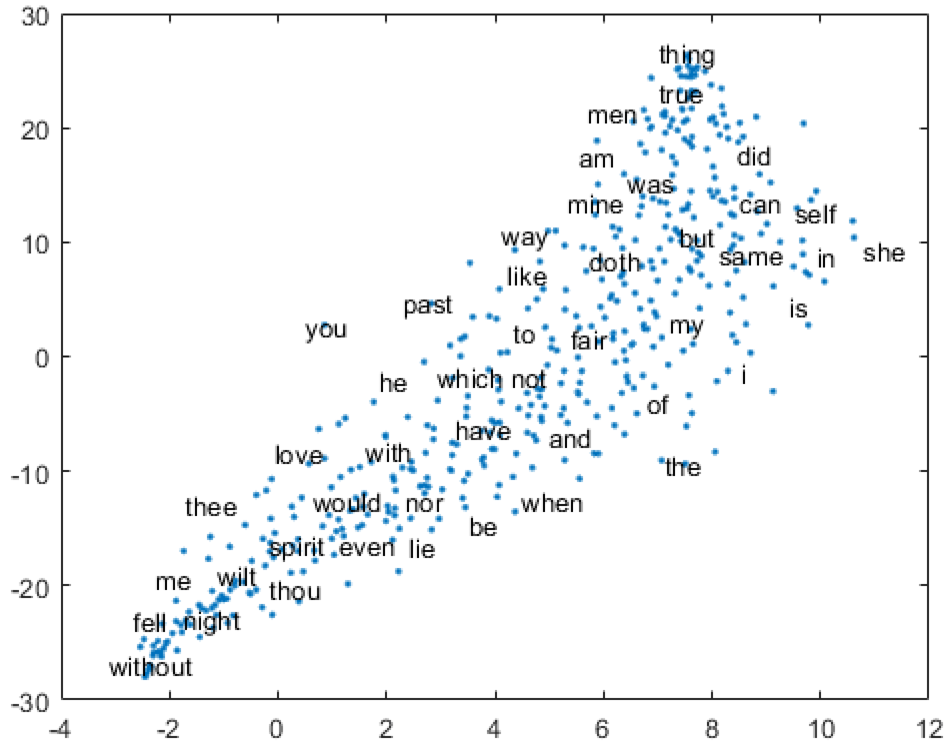
Train a word embedding of dimension 100 using the example text file `exampleSonnetsDocuments.txt`. This file contains preprocessed versions of Shakespeare's sonnets, with one sonnet per line and words separated by a space.

```
filename = "exampleSonnetsDocuments.txt";
emb = trainWordEmbedding(filename)
```

```
Training: 100% Loss: 2.71773  Remaining time: 0 hours 0 minutes.

emb = 
  wordEmbedding with properties:

     Dimension: 100
    Vocabulary: [1x502 string]
```

View the word embedding in a text scatter plot using `tsne`.

```
words = emb.Vocabulary;
V = word2vec(emb,words);
XY = tsne(V);
textscatter(XY,words)
```



Train Word Embedding from Documents

Train a word embedding using the example data `sonnetsPreprocessed.txt`. This file contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

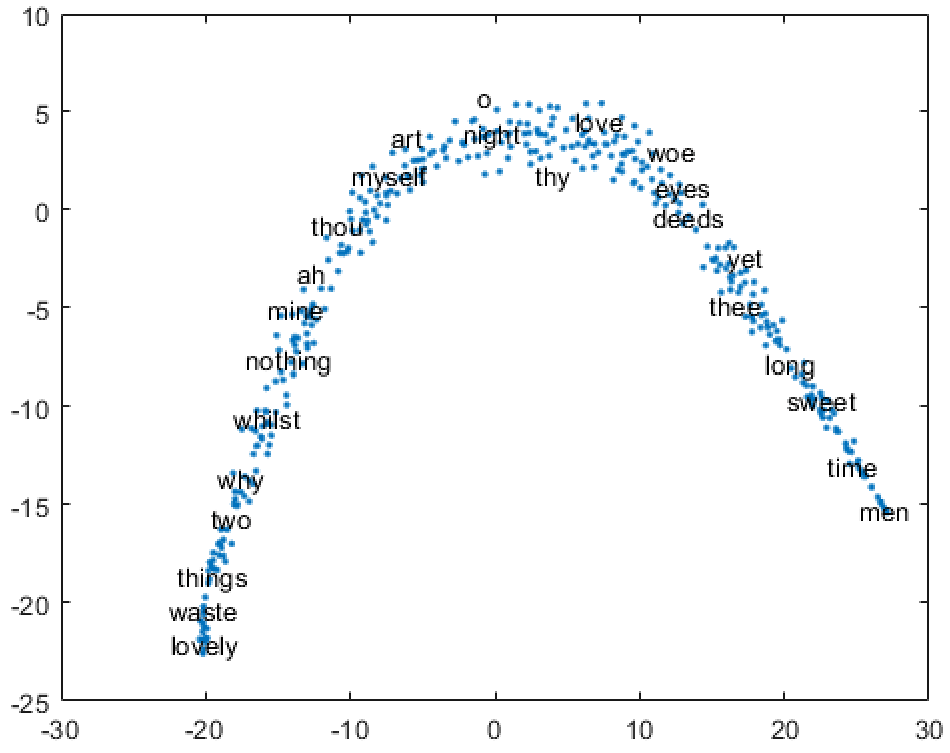
Train a word embedding using `trainWordEmbedding`.

```
emb = trainWordEmbedding(documents)
```

```
Training: 100% Loss: 2.79758  Remaining time: 0 hours 0 minutes.

emb = 
  wordEmbedding with properties:

     Dimension: 100
    Vocabulary: [1x401 string]
```

Visualize the word embedding in a text scatter plot using `tsne`.

```
words = emb.Vocabulary;
V = word2vec(emb,words);
XY = tsne(V);
textscatter(XY,words)
```



Specify Word Embedding Options

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

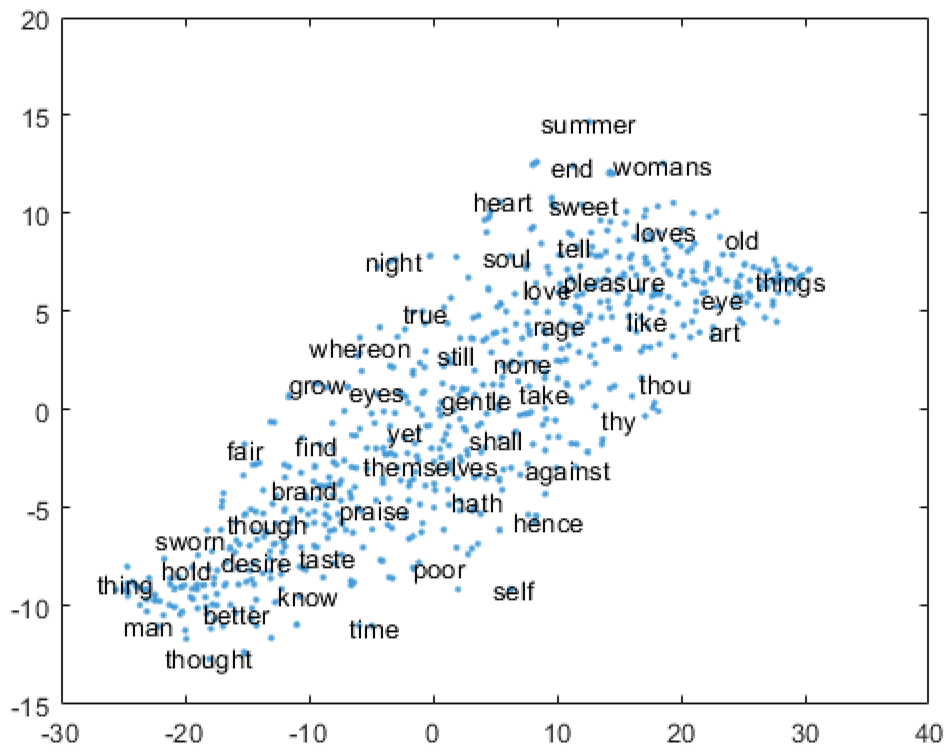
Specify the word embedding dimension to be 50. To reduce the number of words discarded by the model, set `'MinCount'` to 3. To train for longer, set the number of epochs to 10.

```
emb = trainWordEmbedding(documents, ...
    'Dimension',50, ...
    'MinCount',3, ...
    'NumEpochs',10)
```

```
Training: 100% Loss: 2.65267  Remaining time: 0 hours 0 minutes.

emb = 
  wordEmbedding with properties:

     Dimension: 50
    Vocabulary: [1x750 string]
```

View the word embedding in a text scatter plot using `tsne`.

```
words = emb.Vocabulary;
V = word2vec(emb, words);
XY = tsne(V);
textscatter(XY,words)
```



Input Arguments

filename — Name of file
string scalar | character vector

Name of the file, specified as a string scalar or character vector.

Data Types: `string` | `char`

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Dimension',50` specifies the word embedding dimension to be 50.

`Dimension` — Dimension of word embedding
100 (default) | positive integer

Dimension of the word embedding, specified as the comma-separated pair consisting of `'Dimension'` and a nonnegative integer.

Example: 300

`Window` — Size of context window
5 (default) | nonnegative integer

Size of the context window, specified as the comma-separated pair consisting of `'Window'` and a nonnegative integer.

Example: 10

`Model` — Model
`'skipgram'` (default) | `'cbow'`

Model, specified as the comma-separated pair consisting of `'Model'` and `'skipgram'` (skip gram) or `'cbow'` (continuous bag-of-words).

Example: `'cbow'`

`DiscardFactor` — Factor to determine word discard rate
`1e-4` (default) | positive scalar

Factor to determine the word discard rate, specified as the comma-separated pair consisting of `'DiscardFactor'` and a positive scalar. The function discards a word from the input window with probability `1-sqrt(t/f) - t/f` where f is the unigram probability of the word, and `t` is `DiscardFactor`. Usually, `DiscardFactor` is in the range of `1e-3` through `1e-5`.

Example: 0.005

`LossFunction` — Loss function
`'ns'` (default) | `'hs'` | `'softmax'`

Loss function, specified as the comma-separated pair consisting of `'LossFunction'` and `'ns'` (negative sampling), `'hs'` (hierarchical softmax), or `'softmax'` (softmax).

Example: `'hs'`

`NumNegativeSamples` — Number of negative samples
5 (default) | positive integer

Number of negative samples for the negative sampling loss function, specified as the comma-separated pair consisting of `'NumNegativeSamples'` and a positive integer. This option is only valid when `LossFunction` is `'ns'`.

Example: 10

`NumEpochs` — Number of epochs
5 (default) | positive integer

Number of epochs for training, specified as the comma-separated pair consisting of `'NumEpochs'` and a positive integer.

Example: 10

`MinCount` — Minimum count of words
5 (default) | positive integer

Minimum count of words to include in the embedding, specified as the comma-separated pair consisting of `'MinCount'` and a positive integer. The function discards words that appear fewer than `MinCount` times in the training data from the vocabulary.

Example: 10

`NGramRange` — Inclusive range for subword n-grams
`[3 6]` (default) | vector of two nonnegative integers

Inclusive range for subword n-grams, specified as the comma-separated pair consisting of `'NGramRange'` and a vector of two nonnegative integers `[min max]`. If you do not want to use n-grams, then set `'NGramRange'` to `[0 0]`.

Example: `[5 10]`

`InitialLearnRate` — Initial learn rate
0.05 (default) | positive scalar

Initial learn rate, specified as the comma-separated pair consisting of `'InitialLearnRate'` and a positive scalar.

Example: 0.01

`UpdateRate` — Rate for updating learn rate
100 (default) | positive integer

Rate for updating the learn rate, specified as the comma-separated pair consisting of `'UpdateRate'` and a positive integer. The learn rate decreases to zero linearly in steps every *N* words where *N* is the `UpdateRate`.

Example: 50

`Verbose` — Verbosity level
1 (default) | 0

Verbosity level, specified as the comma-separated pair consisting of `'Verbose'` and one of the following:

- 0 – Do not display verbose output.
- 1 – Display progress information.

Example: `'Verbose',0`

Output Arguments

emb — Output word embedding
word embedding

Output word embedding, returned as a `wordEmbedding` object.

More About

Language Considerations

File input to the `trainWordEmbedding` function requires words separated by whitespace.

For files containing non-English text, you might need to input a `tokenizedDocument` array to `trainWordEmbedding`.

To create a `tokenizedDocument` array from pretokenized text, use the `tokenizedDocument` function and set the `'TokenizeMethod'` option to `'none'`.

Tips

The training algorithm uses the number of threads given by the function `maxNumCompThreads`. To learn how to change the number of threads used by MATLAB, see `maxNumCompThreads`.

See Also

`doc2sequence` | `fastTextWordEmbedding` | `readWordEmbedding` | `tokenizedDocument` | `vec2word` | `word2vec` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding` | `writeWordEmbedding`

Topics
“Train a Sentiment Classifier”
“Classify Text Data Using Deep Learning”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2017b

transform

Transform documents into lower-dimensional space

Syntax

```
dscores = transform(lsaMdl,documents)
dscores = transform(lsaMdl,bag)
dscores = transform(lsaMdl,counts)

dscores = transform(ldaMdl,documents)
dscores = transform(ldaMdl,bag)
dscores = transform(ldaMdl,counts)
dscores = transform(___,Name,Value)
```

Description

`dscores = transform(lsaMdl,documents)` transforms documents into the semantic space of the latent semantic analysis (LSA) model `lsaMdl`.

`dscores = transform(lsaMdl,bag)` transforms documents represented by the bag-of-words or bag-of-n-grams model `bag` into the semantic space of the LSA model `lsaMdl`.

`dscores = transform(lsaMdl,counts)` transforms documents represented by the matrix of word counts into the semantic space of the LSA model `lsaMdl`.

`dscores = transform(ldaMdl,documents)` transforms documents into the latent Dirichlet allocation (LDA) topic probability space of LDA model `ldaMdl`. The rows of `dscores` are the topic mixture representations of the documents.

`dscores = transform(ldaMdl,bag)` transforms documents represented by the bag-of-words or bag-of-n-grams model `bag` into the LDA topic probability space of LDA model `ldaMdl`.

`dscores = transform(ldaMdl,counts)` transforms documents represented by the matrix of word counts into the LDA topic probability space of LDA model `ldaMdl`.

`dscores = transform(___,Name,Value)` specifies additional options using one or more name-value pair arguments. These name-value pairs only apply if the input model is an `ldaModel` object.

Examples

Transform Documents into LSA Semantic Space

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
```

```
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LSA model with 20 components.

```
numCompnents = 20;
mdl = fitlsa(bag,numCompnents)
```

```
mdl = 
  lsaModel with properties:

              NumComponents: 20
           ComponentWeights: [1x20 double]
             DocumentScores: [154x20 double]
                 WordScores: [3092x20 double]
                 Vocabulary: [1x3092 string]
    FeatureStrengthExponent: 2
```

Use `transform` to transform the first 10 documents into the semantic space of the LSA model.

```
dscores = transform(mdl,documents(1:10))
```

```
dscores = 10×20

    5.6059   -1.8559    0.9286   -0.7086   -0.4652   -0.8340   -0.6751    0.0611   -0.2268    1.9
    7.3069   -2.3578    1.8359   -2.3442   -1.5776   -2.0310   -0.7948    1.3411    1.1700    1.8
    7.1056   -2.3508   -2.8837   -1.0688   -0.3462   -0.6962   -0.0334   -0.0472   -0.4916    0.6
    8.6292   -3.0471   -0.8512   -0.4356   -0.3055    0.4671    1.4219   -0.8454    0.8270    0.4
    1.0434    1.7490    0.8703   -2.2315   -1.1221    0.2848    2.0522   -0.6975   -1.7191   -0.2
    6.8358   -2.0806   -3.3798   -1.0452   -0.2075    2.0970    0.4477    0.2080   -0.9532    1.6
    2.3847    0.3923   -0.4323   -1.5340    0.4023   -1.0396    1.0326    0.3776   -0.2101   -1.0
    3.7925   -0.3941   -4.4610   -0.4930    0.4651    0.3404    0.5493    0.1470   -0.5065    0.2
    4.6522    0.7188   -1.1787   -0.8996    0.3360    0.4531    0.1935    0.3328    0.8640   -1.6
    8.8218   -0.8168   -2.5101    1.1197   -0.8673   -1.2336   -0.0768    0.1943    0.7629   -0.1
```

Transform Documents into LDA Topic Mixtures

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Fit an LDA model with five topics.

```
numTopics = 5;
mdl = fitlda(bag,numTopics)
```

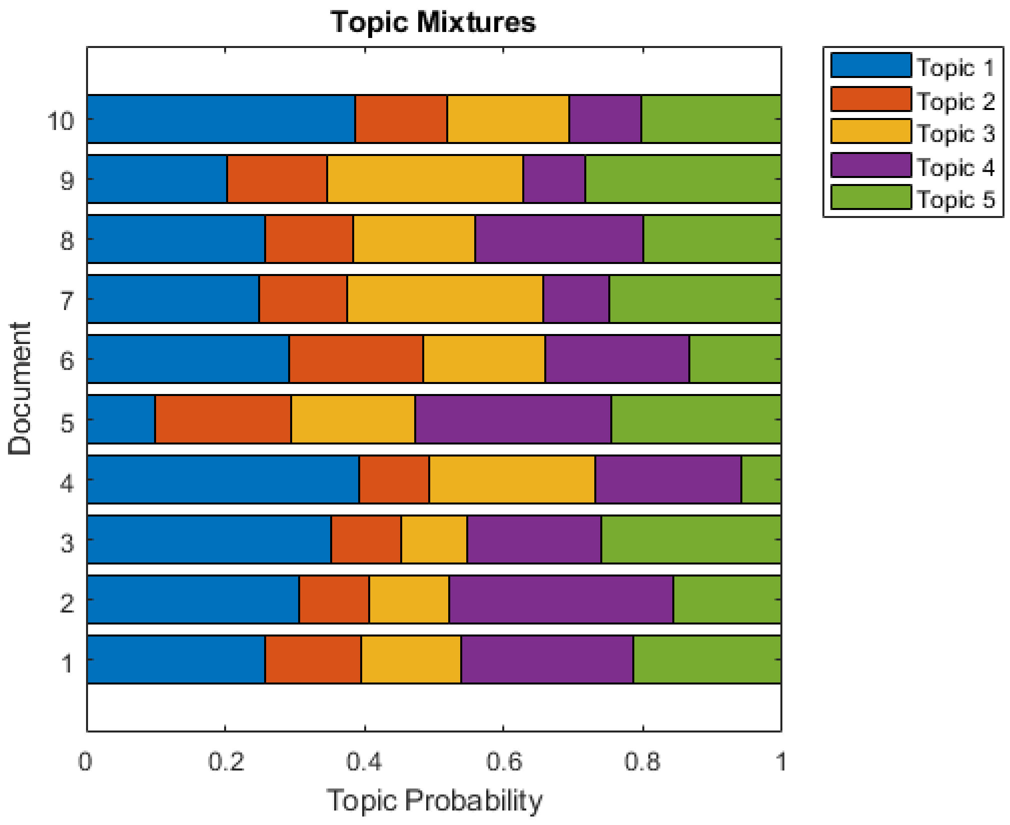
```
Initial topic assignments sampled in 0.0506626 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.00 |            |  1.212e+03 |         1.250 |             0 |
|          1 |       0.02 | 1.2300e-02 |  1.112e+03 |         1.250 |             0 |
|          2 |       0.02 | 1.3254e-03 |  1.102e+03 |         1.250 |             0 |
|          3 |       0.02 | 2.9402e-05 |  1.102e+03 |         1.250 |             0 |
=====================================================================================
```

```
mdl = 
  ldaModel with properties:

                     NumTopics: 5
             WordConcentration: 1
            TopicConcentration: 1.2500
      CorpusTopicProbabilities: [0.2000 0.2000 0.2000 0.2000 0.2000]
    DocumentTopicProbabilities: [154x5 double]
        TopicWordProbabilities: [3092x5 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Use `transform` to transform the documents into a vector of topic probabilities. You can visualize these mixtures using stacked bar charts. View the topic mixtures of the first 10 documents.

```
topicMixtures = transform(mdl,documents(1:10));
figure
```

```
barh(topicMixtures,'stacked')
xlim([0 1])
title("Topic Mixtures")
xlabel("Topic Probability")
ylabel("Document")
legend("Topic " + string(1:numTopics),'Location','northeastoutside')
```



Transform Word Count Matrix into LDA Topic Mixtures

Load the example data. `sonnetsCounts.mat` contains a matrix of word counts and a corresponding vocabulary of preprocessed versions of Shakespeare's sonnets.

```
load sonnetsCounts.mat
size(counts)
```

```
ans = 1×2

         154         3092
```

Fit an LDA model with 20 topics. To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
numTopics = 20;
mdl = fitlda(counts,numTopics)
```

```
Initial topic assignments sampled in 0.0476288 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.01 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.05 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.06 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.05 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.05 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.05 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.05 | 6.4180e-05 |  7.291e+02 |         5.000 |             0 |
=====================================================================================

mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Use `transform` to transform the documents into a vector of topic probabilities.

```
topicMixtures = transform(mdl,counts(1:10,:))
```

```
topicMixtures = 10×20

    0.0167    0.0035    0.1645    0.0977    0.0433    0.0833    0.0987    0.0033    0.0299    0.0
    0.0711    0.0544    0.0116    0.0044    0.0033    0.0033    0.0431    0.0053    0.0145    0.0
    0.0293    0.0482    0.1078    0.0322    0.0036    0.0036    0.0464    0.0036    0.0064    0.0
    0.0055    0.0962    0.2403    0.0033    0.0296    0.1613    0.0164    0.0955    0.0163    0.0
    0.0341    0.0224    0.0341    0.0645    0.0948    0.0038    0.0189    0.1099    0.0187    0.0
    0.0445    0.0035    0.1167    0.0034    0.0446    0.0583    0.1268    0.0169    0.0034    0.1
    0.1720    0.0764    0.0090    0.0180    0.0325    0.1213    0.0036    0.0036    0.0505    0.0
    0.0043    0.0033    0.1248    0.0033    0.0299    0.0033    0.0690    0.1699    0.0695    0.0
    0.0412    0.0387    0.0555    0.0165    0.0166    0.0433    0.0033    0.0038    0.0048    0.0
    0.0362    0.0035    0.1117    0.0304    0.0034    0.1248    0.0439    0.0340    0.0168    0.0
```

Input Arguments

`lsaMdl` — Input LSA model
`lsaModel` object

Input LSA model, specified as an `lsaModel` object.

`ldaMdl` — Input LDA model
`ldaModel` object

Input LDA model, specified as an `ldaModel` object.

`documents` — Input documents
`tokenizedDocument` array | string array of words | cell array of character vectors

Input documents, specified as a `tokenizedDocument` array, a string array of words, or a cell array of character vectors. If `documents` is a `tokenizedDocument`, then it must be a column vector. If `documents` is a string array or a cell array of character vectors, then it must be a row of the words of a single document.

Tip To ensure that the function does not discard useful information, you must first preprocess the input documents using the same steps used to preprocess the documents used to train the model.

bag — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`counts` — Frequency counts of words
matrix of nonnegative integers

Frequency counts of words, specified as a matrix of nonnegative integers. If you specify `'DocumentsIn'` to be `'rows'`, then the value `counts(i,j)` corresponds to the number of times the *j*th word of the vocabulary appears in the *i*th document. Otherwise, the value `counts(i,j)` corresponds to the number of times the *i*th word of the vocabulary appears in the *j*th document.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IterationLimit',200` sets the iteration limit to 200.

Note These name-value pairs only apply if the input model is an `ldaModel` object.

`DocumentsIn` — Orientation of documents
`'rows'` (default) | `'columns'`

Orientation of documents in the word count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Input is a matrix of word counts with rows corresponding to documents.
- `'columns'` – Input is a transposed matrix of word counts with columns corresponding to documents.

This option only applies if you specify the input documents as a matrix of word counts.

Note If you orient your word count matrix so that documents correspond to columns and specify `'DocumentsIn','columns'`, then you might experience a significant reduction in optimization-execution time.

`IterationLimit` — Maximum number of iterations

`100` (default) | positive integer

Maximum number of iterations, specified as the comma-separated pair consisting of `'IterationLimit'` and a positive integer.

Example: `'IterationLimit',200`

`LogLikelihoodTolerance` — Relative tolerance on log-likelihood

`0.0001` (default) | positive scalar

Relative tolerance on log-likelihood, specified as the comma-separated pair consisting of `'LogLikelihoodTolerance'` and a positive scalar. The optimization terminates when this tolerance is reached.

Example: `'LogLikelihoodTolerance',0.001`

Output Arguments

`dscores` — Output document scores

matrix

Output document scores, returned as a matrix of score vectors.

See Also

`bagOfWords` | `fitlda` | `fitlsa` | `ldaModel` | `logp` | `lsaModel` | `predict` | `wordcloud`

Topics

"Analyze Text Data Using Topic Models"
"Prepare Text Data for Analysis"
"Extract Text Data from Files"

Introduced in R2017b

upper

Convert documents to uppercase

Syntax

`newDocuments = upper(documents)`

Description

`newDocuments = upper(documents)` converts each lowercase character in the input documents to the corresponding uppercase character, and leaves all other characters unchanged.

Examples

Convert Documents to Uppercase

Convert all lowercase characters in an array of documents to uppercase.

```
documents = tokenizedDocument([
    "An Example of a Short Sentence"
    "A Second Short Sentence"])
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: An Example of a Short Sentence
    4 tokens: A Second Short Sentence
```

```
newDocuments = upper(documents)
```

```
newDocuments = 
  2x1 tokenizedDocument:

    6 tokens: AN EXAMPLE OF A SHORT SENTENCE
    4 tokens: A SECOND SHORT SENTENCE
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`newDocuments` — Output documents
`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

See Also

`decodeHTMLEntities` | `erasePunctuation` | `eraseTags` | `eraseURLs` | `lower` | `tokenizedDocument`

Topics

"Prepare Text Data for Analysis"
"Create Simple Text Model for Classification"

Introduced in R2017b

vaderSentimentScores

Sentiment scores with VADER algorithm

Syntax

```
compoundScores = vaderSentimentScores(documents)
compoundScores = vaderSentimentScores(documents,Name,Value)
[compoundScores,positiveScores,negativeScores,neutralScores] =
vaderSentimentScores( ___ )
```

Description

Use `vaderSentimentScores` to evaluate sentiment in tokenized text with the Valence Aware Dictionary and sEntiment Reasoner (VADER) algorithm. The `vaderSentimentScores` function uses, by default, the VADER sentiment lexicon and modifier word lists.

The function supports English text only.

`compoundScores = vaderSentimentScores(documents)` returns sentiment scores for tokenized documents. The function calculates the compound scores by aggregating individual token scores, adjusted according to the algorithm rules and then normalized between -1 and 1. The function discards all tokens with a single character, not present in the sentiment lexicon.

`compoundScores = vaderSentimentScores(documents,Name,Value)` specifies additional options using one or more name-value pairs.

`[compoundScores,positiveScores,negativeScores,neutralScores] = vaderSentimentScores( ___ )` also returns the ratios for proportions of the documents which are positive, negative, and neutral, respectively, using any of the previous syntaxes.

Examples

Evaluate Sentiment in Text

Create a tokenized document.

```
str = [
    "The book was VERY good!!!!"
    "The book was not very good."];
documents = tokenizedDocument(str);
```

Evaluate the sentiment of the tokenized documents. Scores close to 1 indicate positive sentiment, scores close to -1 indicate negative sentiment, and scores close to 0 indicate neutral sentiment.

```
compoundScores = vaderSentimentScores(documents)
```

```
compoundScores = 2×1

    0.7264
```

```
   -0.3865
```

Evaluate Sentiment Using Custom Lexicon

Sentiment analysis algorithms such as VADER rely on annotated lists of words called sentiment lexicons. For example, VADER uses a sentiment lexicon with words annotated with a sentiment score ranging from -1 to 1, where scores close to 1 indicate strong positive sentiment, scores close to -1 indicate strong negative sentiment, and scores close to zero indicate neutral sentiment.

To analyze the sentiment of text using the VADER algorithm, use the `vaderSentimentScores` function. If the sentiment lexicon used by the `vaderSentimentScores` function does not suit the data you are analyzing, for example, if you have a domain-specific data set like medical or engineering data, then you can use your own custom sentiment lexicon. For an example showing how to generate a domain specific sentiment lexicon, see “Generate Domain Specific Sentiment Lexicon”.

Create a tokenized document array containing the text data to analyze.

```
textData = [
    "This company is showing extremely strong growth."
    "This other company is accused of misleading consumers."];
documents = tokenizedDocument(textData);
```

Load the example domain specific lexicon for finance data.

```
filename = "financeSentimentLexicon.csv";
tbl = readtable(filename);
head(tbl)
```

```
ans=8×2 table
         Token          SentimentScore
    ________________    ______________

    {'innovative' }            1
    {'efficiency' }      0.91852
    {'strong'     }      0.82362
    {'efficiently'}      0.81475
    {'creative'   }      0.74264
    {'enhance'    }      0.73791
    {'innovations'}      0.72985
    {'improved'   }      0.71476
```

Evaluate the sentiment using the `vaderSentimentScores` function and specify the custom sentiment lexicon using the `'SentimentLexicon'` option. Scores close to 1 indicate positive sentiment, scores close to -1 indicate negative sentiment, and scores close to 0 indicate neutral sentiment.

```
compoundScores = vaderSentimentScores(documents,'SentimentLexicon',tbl)
```

```
compoundScores = 2×1

    0.2834
   -0.1273
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Boosters',["verry" "verrry"]` specifies to use the boosters "verry" and "verrrry".

`SentimentLexicon` — Sentiment lexicon
table

Sentiment lexicon, specified as a table with the following columns:

- `Token` – Token, specified as a string scalar. The tokens must be lowercase.
- `SentimentScore` – Sentiment score of token, specified as a numeric scalar.

When evaluating sentiment, the software, by default, ignores tokens with one character and replaces emojis with an equivalent textual description before computing the sentiment scores. For example, the software replaces instances of the emoji "▯▯" with the text "grinning face" and then evaluates the sentiment scores. If you provide tokens with one character or emojis with corresponding sentiment scores in `SentimentLexicon`, then the function does not remove or replace these tokens.

The default sentiment lexicon is the VADER sentiment lexicon.

Data Types: `table`

`Boosters` — List of booster words or n-grams
string array

List of booster words or n-grams, specified as a string array.

The function uses booster n-grams to boost the sentiment of subsequent tokens. For example, words like "absolutely" and "amazingly".

For a list of words, the list must be a column vector. For a list of n-grams, the list has size `NumNgrams`-by-`maxN` , where `NumNgrams` is the number of n-grams, and `maxN` is the length of the largest n-gram. The `(i,j)`th element of the list is the `j`th word of the `i`th n-gram. If the number of words in the `i`th n-gram is less than `maxN`, then the remaining entries of the `i`th row of the list are empty.

The booster n-grams must be lowercase.

The default list of booster n-grams is the VADER booster list.

Data Types: `string`

`Dampeners` — List of dampener words or n-grams
string array

List of dampener words or n-grams, specified as a string array.

The function uses dampener n-grams to dampen the sentiment of subsequent tokens. For example, words like "hardly" and "somewhat".

For a list of words, the list must be a column vector. For a list of n-grams, the list has size `NumNgrams`-by-`maxN` , where `NumNgrams` is the number of n-grams, and `maxN` is the length of the largest n-gram. The `(i,j)`th element of the list is the `j`th word of the `i`th n-gram. If the number of words in the `i`th n-gram is less than `maxN`, then the remaining entries of the `i`th row of the list are empty.

The dampener n-grams must be lowercase.

The default list of dampener n-grams is the VADER booster list.

Data Types: `string`

`Negations` — List of negation words
string array

List of negation words, specified as a string array.

The function uses negation words to negate the sentiment of subsequent tokens. For example, words like "not" and "isn't".

The negation words must be lowercase.

The default list of negation words is the VADER negation list.

Data Types: `string`

Output Arguments

`compoundScores` — Compound sentiment scores
numeric vector

Compound sentiment scores, returned as a numeric vector. The function returns one score for each input document. The value `compoundScores(i)` corresponds to the compound sentiment score of `documents(i)`.

The function determines the compound scores by aggregating individual token scores, adjusts them according to the VADER algorithm rules, and then normalizes them between -1 and 1.

`positiveScores` — Positive sentiment scores
numeric vector

Positive sentiment scores, returned as a numeric vector. The function returns one score for each input document. The value `positiveScores(i)` corresponds to the positive sentiment score of `documents(i)`.

`negativeScores` — Negative sentiment scores
numeric vector

Negative sentiment scores, returned as a numeric vector. The function returns one score for each input document. The value `negativeScores(i)` corresponds to the negative sentiment score of `documents(i)`.

`neutralScores` — Neutral sentiment scores
numeric vector

Neutral sentiment scores, returned as a numeric vector. The function returns one score for each input document. The value `neutralScores(i)` corresponds to the neutral sentiment score of `documents(i)`.

References

[1] Hutto, Clayton J., and Eric Gilbert. "Vader: A parsimonious rule-based model for sentiment analysis of social media text." In *Eighth international AAAI conference on weblogs and social media*. 2014.

See Also

`ratioSentimentScores` | `tokenizedDocument`

Topics
“Analyze Sentiment in Text”
“Generate Domain Specific Sentiment Lexicon”
“Train a Sentiment Classifier”
“Create Simple Text Model for Classification”
“Analyze Text Data Containing Emojis”
“Analyze Text Data Using Topic Models”

Introduced in R2019b

vec2word

Map embedding vector to word

Syntax

```
words = vec2word(emb,M)
[words,dist] = vec2word(emb,M)
___ = vec2word(emb,M,k)
___ = vec2word(___,'Distance',distance)
```

Description

`words = vec2word(emb,M)` returns the closest words to the embedding vectors in the rows of `M`.

`[words,dist] = vec2word(emb,M)` returns the closest words to the embedding vectors in `M`, and returns the distances `dist` of each to their source vectors.

`___ = vec2word(emb,M,k)` returns the top `k` closest words.

`___ = vec2word(___,'Distance',distance)` specifies the distance metric.

Examples

Map Words to Vectors and Back

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

Map the words "Italy", "Rome", and "Paris" to vectors using `word2vec`.

```
italy = word2vec(emb,"Italy");
rome = word2vec(emb,"Rome");
paris = word2vec(emb,"Paris");
```

Map the vector `italy - rome + paris` to a word using `vec2word`.

```
word = vec2word(emb,italy - rome + paris)
```

```
word = 
"France"
```

Find Closest Words to Vector

Find the top five closest words to a word embedding vector and their distances.

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding;
```

Map the words "Italy", "Rome", and "Paris" to vectors using word2vec.

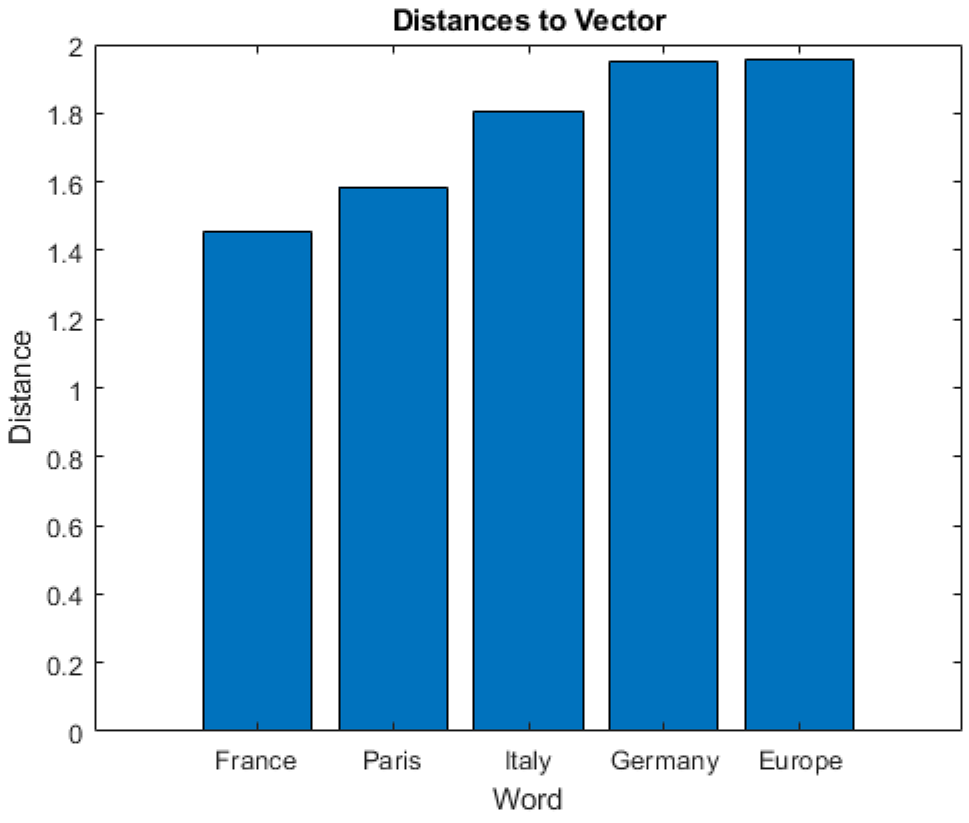
```
italy = word2vec(emb,"Italy");
rome = word2vec(emb,"Rome");
paris = word2vec(emb,"Paris");
```

Map the vector `italy - rome + paris` to a word using `vec2word`. Find the top five closest words using the Euclidean distance metric.

```
k = 5;
M = italy - rome + paris;
[words,dist] = vec2word(emb,M,k,'Distance','euclidean');
```

Plot the words and distances in a bar chart.

```
figure;
bar(dist)
xticklabels(words)
xlabel("Word")
ylabel("Distance")
title("Distances to Vector")
```



Input Arguments

emb — Input word embedding
`wordEmbedding` object

Input word embedding, specified as a `wordEmbedding` object.

`M` — Word embedding vectors
matrix

Word embedding vectors, specified as a matrix. Each row of `M` is a word embedding vector. `M` must have `emb.Dimension` columns.

k — Number of closest words
positive integer

Number of closest words to return, specified as a positive integer.

`distance` — Distance metric
`'cosine'` (default) | `'euclidean'`

Distance metric, specified as `'cosine'` or `'euclidean'`.

Output Arguments

`words` — Output words
string vector

Output words, returned as a string vector.

`dist` — Distance of words to source vectors
vector

Distance of words to their source vectors, returned as a vector.

See Also

`doc2sequence` | `fastTextWordEmbedding` | `ind2word` | `isVocabularyWord` | `tokenizedDocument` | `word2ind` | `word2vec` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding`

Topics
"Train a Sentiment Classifier"
"Classify Text Data Using Deep Learning"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2017b

word2ind

Map word to encoding index

Syntax

```
M = word2ind(enc,words)
M = word2ind(enc,words,'IgnoreCase',true)
```

Description

`M = word2ind(enc,words)` returns the indices of `words` in the encoding `enc`. The function, by default, is case sensitive.

`M = word2ind(enc,words,'IgnoreCase',true)` returns indices ignoring case using any of the previous syntaxes. If multiple words in the encoding differ only in case, then the function returns the index corresponding to one of them and does not return any particular index.

Examples

Map Words to Encoding Indices

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
documents(1:10)
```

```
ans = 
  10x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youth
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair th
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial t
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lovs
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt hap
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art belo
```

Create a word encoding.

```
enc = wordEncoding(documents)
enc = 
  wordEncoding with properties:

      NumWords: 3092
    Vocabulary: [1x3092 string]
```

Map the words "rose", "love", and "beauty" to encoding indices using the `word2ind` function.

```
words = ["rose" "love" "beauty"];
idx = word2ind(enc,words)
```

```
idx = 1×3

     7   387    79
```

Input Arguments

enc — Input word encoding
`wordEncoding` object

Input word encoding, specified as a `wordEncoding` object.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

Output Arguments

`M` — Vector of word encoding indices
vector of positive integers

Vector of word encoding indices.

See Also

`fastTextWordEmbedding` | `ind2word` | `isVocabularyWord` | `tokenizedDocument` | `word2vec` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding`

Topics
“Train a Sentiment Classifier”
“Classify Text Data Using Deep Learning”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2018b

word2vec

Map word to embedding vector

Syntax

```
M = word2vec(emb,words)
M = word2vec(emb,words,'IgnoreCase',true)
```

Description

`M = word2vec(emb,words)` returns the embedding vectors of `words` in the embedding `emb`. If a word is not in the embedding vocabulary, then the function returns a row of `NaN`s. The function, by default, is case sensitive.

`M = word2vec(emb,words,'IgnoreCase',true)` returns the embedding vectors of `words` ignoring case using any of the previous syntaxes. If multiple words in the embedding differ only in case, then the function returns the vector corresponding to one of them and does not return any particular vector.

Examples

Map Words to Vectors and Back

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

Map the words "Italy", "Rome", and "Paris" to vectors using `word2vec`.

```
italy = word2vec(emb,"Italy");
rome = word2vec(emb,"Rome");
paris = word2vec(emb,"Paris");
```

Map the vector `italy - rome + paris` to a word using `vec2word`.

```
word = vec2word(emb,italy - rome + paris)
```

```
word = 
"France"
```

Input Arguments

`emb` — Input word embedding
`wordEmbedding` object

Input word embedding, specified as a `wordEmbedding` object.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

Output Arguments

`M` — Matrix of word embedding vectors
matrix

Matrix of word embedding vectors.

See Also

`doc2sequence` | `fastTextWordEmbedding` | `isVocabularyWord` | `tokenizedDocument` | `vec2word` | `word2ind` | `wordEmbedding` | `wordEncoding`

Topics
"Train a Sentiment Classifier"
"Classify Text Data Using Deep Learning"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2017b

wordcloud

Create word cloud chart from text, bag-of-words model, bag-of-n-grams model, or LDA model

Syntax

```
wordcloud(str)
wordcloud(documents)
wordcloud(bag)

wordcloud(tbl,wordVar,sizeVar)
wordcloud(words,sizeData)
wordcloud(C)

wordcloud(ldaMdl,topicIdx)

wordcloud( ___ ,Name,Value)

wordcloud(parent, ___ )

wc = wordcloud( ___ )
```

Description

Text Analytics Toolbox extends the functionality of the `wordcloud` (MATLAB) function. It adds support for creating word clouds directly from string arrays, and creating word clouds from bag-of-words models, bag-of-n-gram models, and LDA topics. If you do not have Text Analytics Toolbox installed, then see `wordcloud`.

`wordcloud(str)` creates a word cloud chart by tokenizing and preprocessing the text in `str`, and then displaying the words with sizes corresponding to the word frequency counts. This syntax supports English, Japanese, German, and Korean text.

`wordcloud(documents)` creates a word cloud chart from the words appearing in `documents`.

`wordcloud(bag)` creates a word cloud chart from the bag-of-words or bag-of-n-grams model `bag`.

`wordcloud(tbl,wordVar,sizeVar)` creates a word cloud chart from the table `tbl`. The variables `wordVar` and `sizeVar` in the table specify the words and word sizes respectively.

`wordcloud(words,sizeData)` creates a word cloud chart from elements of `words` with word sizes specified by `sizeData`.

`wordcloud(C)` creates a word cloud chart from the elements of categorical array `C` using frequency counts.

`wordcloud(ldaMdl,topicIdx)` creates a word cloud chart from the topic with index `topicIdx` of the LDA model `ldaMdl`.

`wordcloud( ___ ,Name,Value)` specifies additional `WordCloudChart` properties using one or more name-value pair arguments.

`wordcloud(parent, ___ )` creates the word cloud in the figure, panel, or tab specified by `parent`.

`wc = wordcloud( ___ )` returns the `WordCloudChart` object. Use `wc` to modify properties of the word cloud after creating it. For a list of properties, see WordCloudChart Properties.

Examples

Create Word Cloud from Text Data

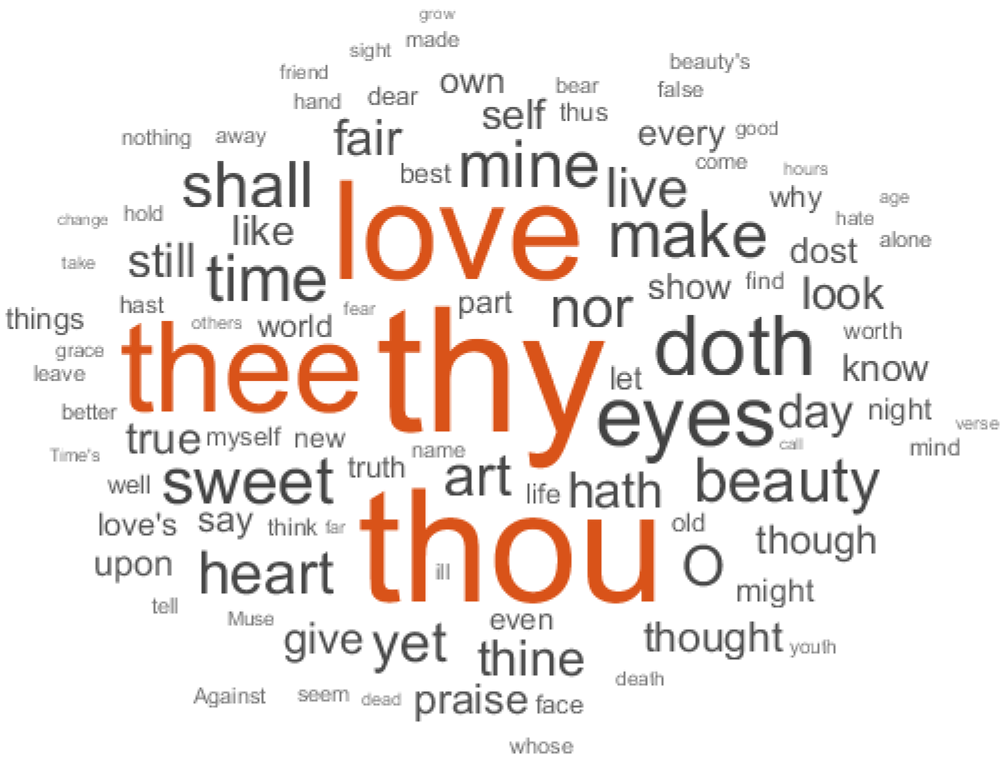
Extract the text from `sonnets.txt` using `extractFileText` and display the text of the first sonnet.

```
str = extractFileText("sonnets.txt");
extractBefore(str,"II")
```

```
ans = 
    "THE SONNETS

     by William Shakespeare




       I

       From fairest creatures we desire increase,
       That thereby beauty's rose might never die,
       But as the riper should by time decease,
       His tender heir might bear his memory:
       But thou, contracted to thine own bright eyes,
       Feed'st thy light's flame with self-substantial fuel,
       Making a famine where abundance lies,
       Thy self thy foe, to thy sweet self too cruel:
       Thou that art now the world's fresh ornament,
       And only herald to the gaudy spring,
       Within thine own bud buriest thy content,
       And tender churl mak'st waste in niggarding:
         Pity the world, or else this glutton be,
         To eat the world's due, by the grave and thee.

       "
```

Display the words from the sonnets in a word cloud.

```
figure
wordcloud(str);
```



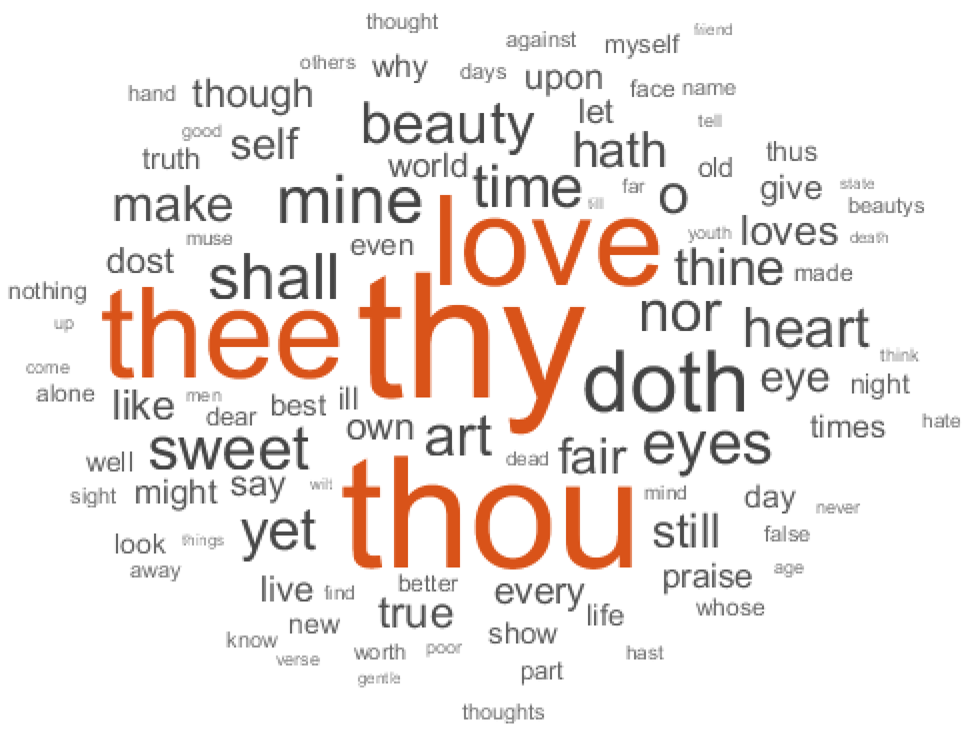
Create Word Cloud from Tokenized Documents

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Visualize the documents using a word cloud.

```
figure
wordcloud(documents);
```



Create Word Cloud from Bag-of-Words Model

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

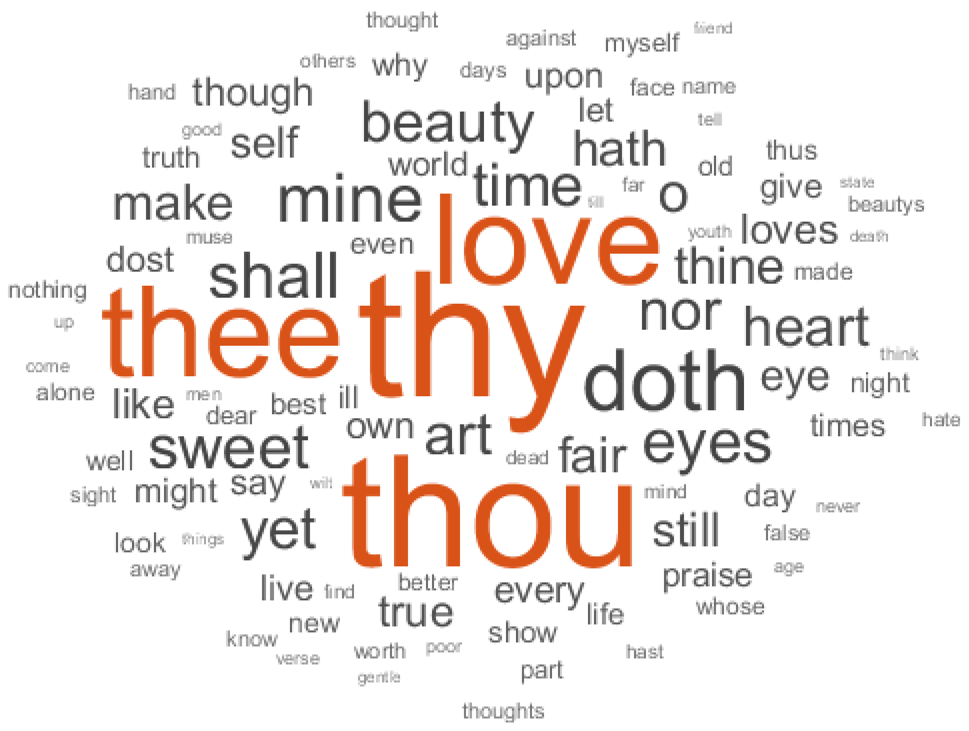
Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

Visualize the bag-of-words model using a word cloud.

```
figure
wordcloud(bag);
```



Create Word Cloud from Table

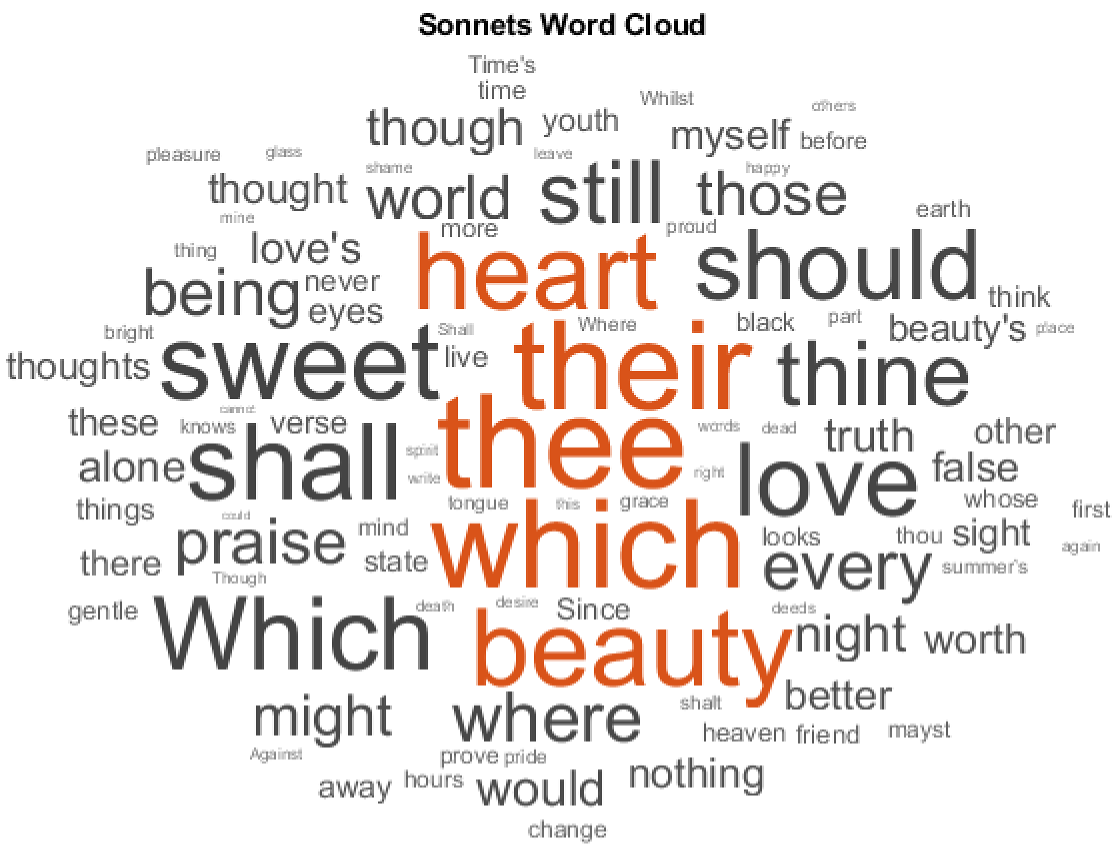
Load the example data `sonnetsTable`. The table `tbl` contains a list of words in the variable `Word`, and the corresponding frequency counts in the variable `Count`.

```
load sonnetsTable
head(tbl)
```

```
ans=8×2 table
        Word         Count
    ___________      _____

    {'''tis'  }        1
    {''Amen'' }        1
    {''Fair'  }        2
    {''Gainst'}        1
    {''Since' }        1
    {''This'  }        2
    {''Thou'  }        1
    {''Thus'  }        1
```

Plot the table data using `wordcloud`. Specify the words and corresponding word sizes to be the `Word` and `Count` variables respectively.

```
figure
wordcloud(tbl,'Word','Count');
title("Sonnets Word Cloud")
```



Create Word Cloud from LDA Topic

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Create a bag-of-words model using `bagOfWords`.

```
bag = bagOfWords(documents)
```

```
bag = 
  bagOfWords with properties:

          Counts: [154x3092 double]
      Vocabulary: [1x3092 string]
        NumWords: 3092
    NumDocuments: 154
```

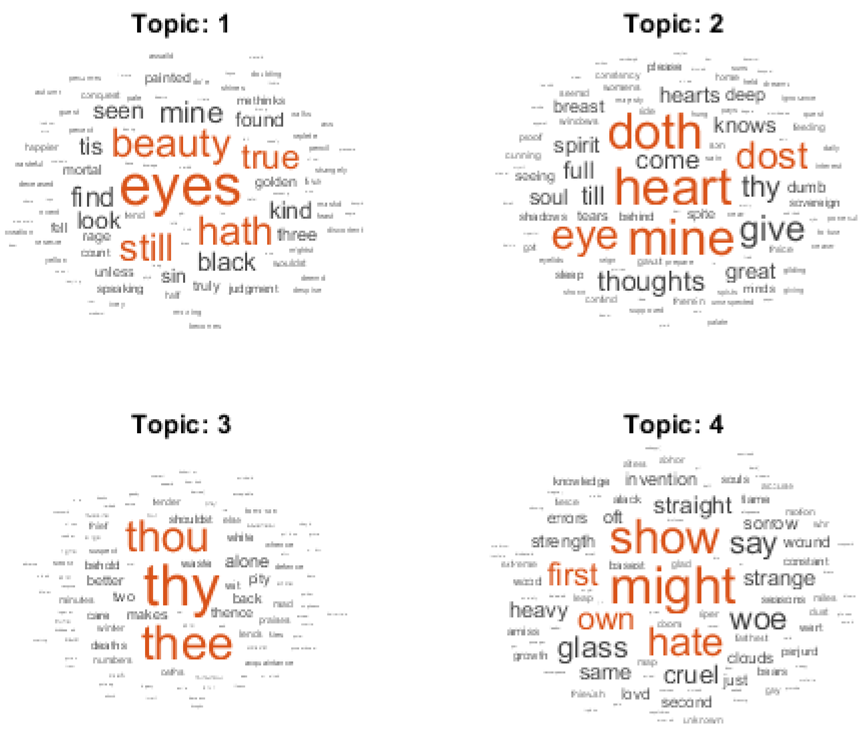
Fit an LDA model with 20 topics. To suppress verbose output, set `'Verbose'` to 0.

```
mdl = fitlda(bag,20,'Verbose',0)
```

```
mdl = 
  ldaModel with properties:

                     NumTopics: 20
             WordConcentration: 1
            TopicConcentration: 5
      CorpusTopicProbabilities: [1x20 double]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
                    Vocabulary: [1x3092 string]
                    TopicOrder: 'initial-fit-probability'
                       FitInfo: [1x1 struct]
```

Visualize the first four topics using word clouds.

```
figure
for topicIdx = 1:4
    subplot(2,2,topicIdx)
    wordcloud(mdl,topicIdx);
    title("Topic: " + topicIdx)
end
```



Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

For string input, the `wordcloud` and `wordCloudCounts` functions use English, Japanese, German, and Korean tokenization, stop word removal, and word normalization.

Example: `["an example of a short document";"a second short document"]`

Data Types: `string` | `char` | `cell`

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`tbl` — Input table
table

Input table, with columns specifying the words and word sizes. Specify the words and the corresponding word sizes in the variables given by `wordVar` and `sizeVar` input arguments respectively.

Data Types: `table`

`wordVar` — Table variable for word data
string scalar | character vector | numeric index | logical vector

Table variable for word data, specified as a string scalar, character vector, numeric index, or a logical vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

`sizeVar` — Table variable for size data
string scalar | character vector | numeric index | logical vector

Table variable for size data, specified as a string scalar, character vector, numeric index, or a logical vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

`words` — Input words
string vector | cell array of character vectors

Input words, specified as a string vector or cell array of character vectors.

Data Types: `string` | `cell`

`sizeData` — Word size data
numeric vector

Word size data, specified as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`C` — Input categorical data
categorical array

Input categorical data, specified as a categorical array. The function plots each unique element of `C` with size corresponding to `histcounts(C)`.

Data Types: `categorical`

`bag` — Input model
`bagOfWords` object | `bagOfNgrams` object

Input bag-of-words or bag-of-n-grams model, specified as a `bagOfWords` object or a `bagOfNgrams` object. If `bag` is a `bagOfNgrams` object, then the function treats each n-gram as a single word.

`ldaMdl` — Input LDA model
`ldaModel` object

Input LDA model, specified as an `ldaModel` object.

`topicIdx` — Index of LDA topic
nonnegative integer

Index of LDA topic, specified as a nonnegative integer.

`parent` — Parent
figure | panel | tab

Parent specified as a figure, panel, or tab.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'HighlightColor','blue'` specifies the highlight color to be blue.

The `WordCloudChart` properties listed here are only a subset. For a complete list, see WordCloudChart Properties.

`MaxDisplayWords` — Maximum number of words to display
100 (default) | nonnegative integer

Maximum number of words to display, specified as a non-negative integer. The software displays the `MaxDisplayWords` largest words.

`Color` — Word color
`[0.2510 0.2510 0.2510]` (default) | RGB triplet | character vector containing a color name | matrix

Word color, specified as an RGB triplet, a character vector containing a color name, or an `N`-by-3 matrix where `N` is the length of `WordData`. If `Color` is a matrix, then each row corresponds to an RGB triplet for the corresponding word in `WordData`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	🟥
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	🟩
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	🟦
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	🟦
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	🟪
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	🟨
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	⬛

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	⬜

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	🟦
`[0.8500 0.3250 0.0980]`	`'#D95319'`	🟧
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	🟨
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	🟪
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	🟩
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	🟦
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	🟥

Example: `'blue'`

Example: `[0 0 1]`

`HighlightColor` — Word highlight color

`[0.8510 0.3255 0.0980]` (default) | RGB triplet | character vector containing a color name

Word highlight color, specified as an RGB triplet, or a character vector containing a color name. The software highlights the largest words with this color.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	🟥
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	🟩
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	🟦
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	🟦
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	🟪
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	🟨
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	⬛

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	⬜

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	🟦
`[0.8500 0.3250 0.0980]`	`'#D95319'`	🟧
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	🟨
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	🟪
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	🟩
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	🟦
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Shape — Shape of word cloud
`'oval'` (default) | `'rectangle'`

Shape of word cloud chart, specified as `'oval'` or `'rectangle'`.

Example: `'rectangle'`

Output Arguments

`wc` — `WordCloudChart` object
`WordCloudChart` object

`WordCloudChart` object. You can modify the properties of a `WordCloudChart` after it is created. For more information, see WordCloudChart Properties.

More About

Language Considerations

For string input, the `wordcloud` and `wordCloudCounts` functions use English, Japanese, German, and Korean tokenization, stop word removal, and word normalization.

For other languages, you might need to manually preprocess your text data and specify unique words and corresponding sizes in `wordcloud`.

To specify word sizes in `wordcloud`, input your data as a table or arrays containing the unique words and corresponding sizes.

See Also

`bagOfNgrams` | `bagOfWords` | `textscatter` | `textscatter3` | `tokenizedDocument` | `wordCloudCounts`

Topics

"Visualize Text Data Using Word Clouds"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2017b

wordCloudCounts

Count words for word cloud creation

Syntax

`T = wordCloudCounts(str)`

Description

`T = wordCloudCounts(str)` tokenizes and preprocesses the text in `str` for word cloud creation and returns a table `T` of words and frequency counts. The function supports English, Japanese, German, and Korean text.

Examples

Word Cloud Frequency Counts

Extract the text from `sonnets.txt` using `extractFileText`.

```
str = extractFileText("sonnets.txt");
```

View the first sonnet.

```
i = strfind(str,"I");
ii = strfind(str,"II");
start = i(1);
fin = ii(1);
extractBetween(str,start,fin-1)
```

```
ans = 
    "I
    
       From fairest creatures we desire increase,
       That thereby beauty's rose might never die,
       But as the riper should by time decease,
       His tender heir might bear his memory:
       But thou, contracted to thine own bright eyes,
       Feed'st thy light's flame with self-substantial fuel,
       Making a famine where abundance lies,
       Thy self thy foe, to thy sweet self too cruel:
       Thou that art now the world's fresh ornament,
       And only herald to the gaudy spring,
       Within thine own bud buriest thy content,
       And tender churl mak'st waste in niggarding:
         Pity the world, or else this glutton be,
         To eat the world's due, by the grave and thee.
    
       "
```

Tokenize and preprocess the sonnets text and create a table of word frequency counts.

```
T = wordCloudCounts(str);
head(T)

ans=8×2 table
     Word     Count
    ______    _____

    "thy"      281 
    "thou"     235 
    "love"     188 
    "thee"     162 
    "eyes"      90 
    "doth"      88 
    "make"      63 
    "mine"      63 
```

Input Arguments

`str` — Input text
string array | character vector | cell array of character vectors

Input text, specified as a string array, character vector, or cell array of character vectors.

For string input, the `wordcloud` and `wordCloudCounts` functions use English, Japanese, German, and Korean tokenization, stop word removal, and word normalization.

Example: `["an example of a short document";"a second short document"]`

Data Types: `string` | `char` | `cell`

Output Arguments

T — Table of word counts
table

Table of words counts sorted in order of importance. The table has columns:

`Word`	String scalar of the word.
`Count`	The number of times the word appears in the documents. The function groups the counts of words that differ only by case or have a common stem according to `normalizeWords`. For example, the function groups the counts for "walk", "Walking", "walking", and "walks".

More About

Language Considerations

For string input, the `wordcloud` and `wordCloudCounts` functions use English, Japanese, German, and Korean tokenization, stop word removal, and word normalization.

See Also

`bagOfNgrams` | `bagOfWords` | `textscatter` | `textscatter3` | `tokenizedDocument` | `wordcloud`

Topics
"Visualize Text Data Using Word Clouds"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2017b

wordEmbedding

Word embedding model to map words to vectors and back

Description

A word embedding, popularized by the word2vec, GloVe, and fastText libraries, maps words in a vocabulary to real vectors.

The vectors attempt to capture the semantics of the words, so that similar words have similar vectors. Some embeddings also capture relationships between words, such as "*king is to queen as man is to woman*". In vector form, this relationship is *king* – *man* + *woman* = *queen*.

Creation

Create a word embedding by loading a pretrained embedding using `fastTextWordEmbedding`, reading an embedding from a file using `readWordEmbedding`, or by training an embedding using `trainWordEmbedding`.

Properties

`Dimension` — Dimension of word embedding
positive integer

Dimension of the word embedding, specified as a positive integer.

Example: 300

`Vocabulary` — Unique words in model
string vector

Unique words in the model, specified as a string vector.

Data Types: `string`

Object Functions

vec2word	Map embedding vector to word
word2vec	Map word to embedding vector
isVocabularyWord	Test if word is member of word embedding or encoding
writeWordEmbedding	Write word embedding file

Examples

Download fastText Support Package

Download and install the Text Analytics Toolbox Model *for fastText English 16 Billion Token Word Embedding* support package.

Type `fastTextWordEmbedding` at the command line.

```
fastTextWordEmbedding
```

If the Text Analytics Toolbox Model *for fastText English 16 Billion Token Word Embedding* support package is not installed, then the function provides a link to the required support package in the Add-On Explorer. To install the support package, click the link, and then click **Install**. Check that the installation is successful by typing `emb = fastTextWordEmbedding` at the command line.

```
emb = fastTextWordEmbedding
```

```
emb = 

  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

If the required support package is installed, then the function returns a `wordEmbedding` object.

Map Words to Vectors and Back

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

Map the words "Italy", "Rome", and "Paris" to vectors using `word2vec`.

```
italy = word2vec(emb,"Italy");
rome = word2vec(emb,"Rome");
paris = word2vec(emb,"Paris");
```

Map the vector `italy - rome + paris` to a word using `vec2word`.

```
word = vec2word(emb,italy - rome + paris)
```

```
word = 
"France"
```

Convert Documents to Sequences of Word Vectors

Convert an array of tokenized documents to sequences of word vectors using a pretrained word embedding.

Load a pretrained word embedding using the `fastTextWordEmbedding` function. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding;
```

Load the factory reports data and create a `tokenizedDocument` array.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
textData = data.Description;
documents = tokenizedDocument(textData);
```

Convert the documents to sequences of word vectors using `doc2sequence`. The `doc2sequence` function, by default, left-pads the sequences to have the same length. When converting large collections of documents using a high-dimensional word embedding, padding can require large amounts of memory. To prevent the function from padding the data, set the `'PaddingDirection'` option to `'none'`. Alternatively, you can control the amount of padding using the `'Length'` option.

```
sequences = doc2sequence(emb,documents,'PaddingDirection','none');
```

View the sizes of the first 10 sequences. Each sequence is *D*-by-*S* matrix, where *D* is the embedding dimension, and *S* is the number of word vectors in the sequence.

```
sequences(1:10)
```

```
ans=10×1 cell array
    {300×10 single}
    {300×11 single}
    {300×11 single}
    {300×6  single}
    {300×5  single}
    {300×10 single}
    {300×8  single}
    {300×9  single}
    {300×7  single}
    {300×13 single}
```

Read Word Embedding from Text File

Read the example word embedding. This model was derived by analyzing text from Wikipedia.

```
filename = "exampleWordEmbedding.vec";
emb = readWordEmbedding(filename)
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 50
    Vocabulary: [1x9999 string]
```

Explore the word embedding using `word2vec` and `vec2word`.

```
king = word2vec(emb,"king");
man = word2vec(emb,"man");
```

```
woman = word2vec(emb,"woman");
word = vec2word(emb,king - man + woman)
```

```
word = 
"queen"
```

Write Word Embedding to File

Train a word embedding and write it to a text file.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Train a word embedding using `trainWordEmbedding`.

```
emb = trainWordEmbedding(documents)
```

```
Training: 100% Loss: 2.88661  Remaining time: 0 hours 0 minutes.

emb = 
  wordEmbedding with properties:

     Dimension: 100
    Vocabulary: [1x401 string]
```

Write the word embedding to a text file.

```
filename = "exampleSonnetsEmbedding.vec";
writeWordEmbedding(emb,filename)
```

Read the word embedding file using `readWordEmbedding`.

```
emb = readWordEmbedding(filename)
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 100
    Vocabulary: [1x401 string]
```

See Also

`doc2sequence` | `fastTextWordEmbedding` | `tokenizedDocument` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbeddingLayer` | `wordEncoding`

Topics

"Train a Sentiment Classifier"
"Classify Text Data Using Deep Learning"

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2017b

wordEmbeddingLayer

Word embedding layer for deep learning networks

Description

A word embedding layer maps word indices to vectors.

Use a word embedding layer in a deep learning long short-term memory (LSTM) network. An LSTM network is a type of recurrent neural network (RNN) that can learn long-term dependencies between time steps of sequence data. A word embedding layer maps a sequence of word indices to embedding vectors and learns the word embedding during training.

This layer requires Deep Learning Toolbox™.

Creation

Syntax

```
layer = wordEmbeddingLayer(dimension,numWords)
layer = wordEmbeddingLayer(dimension,numWords,Name,Value)
```

Description

`layer = wordEmbeddingLayer(dimension,numWords)` creates a word embedding layer and specifies the embedding dimension and vocabulary size.

`layer = wordEmbeddingLayer(dimension,numWords,Name,Value)` sets optional properties on page 1-496 using one or more name-value pairs. Enclose each property name in single quotes.

Properties

Word Embedding

`Dimension` — Dimension of word embedding
positive integer

Dimension of the word embedding, specified as a positive integer.

Example: 300

`NumWords` — Number of words in model
positive integer

Number of words in the model, specified as a positive integer. If the number of unique words in the training data is greater than `NumWords`, then the layer maps the out-of-vocabulary words to the same vector.

Parameters and Initialization

`WeightsInitializer` — Function to initialize weights

`'narrow-normal'` (default) | `'glorot'` | `'he'` | `'orthogonal'` | `'zeros'` | `'ones'` | function handle

Function to initialize the weights, specified as one of the following:

- `'narrow-normal'` – Initialize the weights by independently sampling from a normal distribution with zero mean and standard deviation 0.01.
- `'glorot'` – Initialize the weights with the Glorot initializer [1] (also known as Xavier initializer). The Glorot initializer independently samples from a uniform distribution with zero mean and variance `2/(numIn + numOut)`, where `numIn = NumWords + 1` and `numOut = Dimension`.
- `'he'` – Initialize the weights with the He initializer [2]. The He initializer samples from a normal distribution with zero mean and variance `2/numIn`, where `numIn = NumWords + 1`.
- `'orthogonal'` – Initialize the input weights with *Q*, the orthogonal matrix given by the QR decomposition of *Z* = *QR* for a random matrix *Z* sampled from a unit normal distribution. [3]
- `'zeros'` – Initialize the weights with zeros.
- `'ones'` – Initialize the weights with ones.
- Function handle – Initialize the weights with a custom function. If you specify a function handle, then the function must be of the form `weights = func(sz)`, where `sz` is the size of the weights.

The layer only initializes the weights when the `Weights` property is empty.

Data Types: `char` | `string` | `function_handle`

`Weights` — Layer weights

matrix

Layer weights, specified as a `Dimension`-by-(`NumWords`+1) array.

For input integers `i` less than or equal to `NumWords`, the layer outputs the vector `Weights(:,i)`. Otherwise, the layer maps outputs the vector `Weights(:,NumWords+1)`.

Learn Rate and Regularization

`WeightLearnRateFactor` — Learning rate factor for weights

1 (default) | nonnegative scalar

Learning rate factor for the weights, specified as a nonnegative scalar.

The software multiplies this factor by the global learning rate to determine the learning rate for the weights in this layer. For example, if `WeightLearnRateFactor` is 2, then the learning rate for the weights in this layer is twice the current global learning rate. The software determines the global learning rate based on the settings specified with the `trainingOptions` function.

Example: 2

`WeightL2Factor` — L2 regularization factor for weights

1 (default) | nonnegative scalar

L2 regularization factor for the weights, specified as a nonnegative scalar.

The software multiplies this factor by the global L2 regularization factor to determine the L2 regularization for the weights in this layer. For example, if `WeightL2Factor` is 2, then the L2 regularization for the weights in this layer is twice the global L2 regularization factor. You can specify the global L2 regularization factor using the `trainingOptions` function.

Example: 2

Layer

Name — Layer name
`''` (default) | character vector | string scalar

Layer name, specified as a character vector or a string scalar. If `Name` is set to `''`, then the software automatically assigns a name at training time.

Data Types: `char` | `string`

`NumInputs` — Number of inputs
1 (default)

Number of inputs of the layer. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

Input names of the layer. This layer accepts a single input only.

Data Types: `cell`

`NumOutputs` — Number of outputs
1 (default)

Number of outputs of the layer. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

Output names of the layer. This layer has a single output only.

Data Types: `cell`

Examples

Create Word Embedding Layer

Create a word embedding layer with embedding dimension 300 and 5000 words.

```
layer = wordEmbeddingLayer(300,5000)
```

```
layer = 
  WordEmbeddingLayer with properties:

         Name: ''

   Hyperparameters
```

```
    Dimension: 300
     NumWords: 5000

   Learnable Parameters
      Weights: []

  Show all properties
```

Include a word embedding layer in an LSTM network.

```
inputSize = 1;
embeddingDimension = 300;
numWords = 5000;
numHiddenUnits = 200;
numClasses = 10;

layers = [
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(embeddingDimension,numWords)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer]
```

```
layers = 
  6x1 Layer array with layers:

     1   ''   Sequence Input          Sequence input with 1 dimensions
     2   ''   Word Embedding Layer    Word embedding layer with 300 dimensions and 5000 unique wo
     3   ''   LSTM                    LSTM with 200 hidden units
     4   ''   Fully Connected         10 fully connected layer
     5   ''   Softmax                 softmax
     6   ''   Classification Output   crossentropyex
```

Initialize Word Embedding Layer with Pretrained Word Embedding

To initialize a word embedding layer in a deep learning network with the weights from a pretrained word embedding, use the `word2vec` function to extract the layer weights and set the `'Weights'` name-value pair of the `wordEmbeddingLayer` function. The word embedding layer expects columns of word vectors, so you must transpose the output of the `word2vec` function.

```
emb = fastTextWordEmbedding;

words = emb.Vocabulary;
dimension = emb.Dimension;
numWords = numel(words);

layer = wordEmbeddingLayer(dimension,numWords,...
    'Weights',word2vec(emb,words)')
```

```
layer = 
  WordEmbeddingLayer with properties:

         Name: ''
```

```
   Hyperparameters
    Dimension: 300
     NumWords: 999994

   Learnable Parameters
      Weights: [300×999994 single]

  Show all properties
```

To create the corresponding word encoding from the word embedding, input the word embedding vocabulary to the `wordEncoding` function as a list of words.

```
enc = wordEncoding(words)
```

```
enc = 
  wordEncoding with properties:

      NumWords: 999994
    Vocabulary: [1×999994 string]
```

References

[1] Glorot, Xavier, and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–356. Sardinia, Italy: AISTATS, 2010.

[2] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In *Proceedings of the 2015 IEEE International Conference on Computer Vision*, 1026–1034. Washington, DC: IEEE Computer Vision Society, 2015.

[3] Saxe, Andrew M., James L. McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks." *arXiv preprint arXiv:1312.6120* (2013).

Extended Capabilities

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`doc2sequence` | `fastTextWordEmbedding` | `lstmLayer` | `sequenceInputLayer` | `tokenizedDocument` | `trainNetwork` | `trainWordEmbedding` | `word2vec` | `wordEncoding`

Topics
“Train a Sentiment Classifier”
“Classify Text Data Using Deep Learning”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2018b

wordEncoding

Word encoding model to map words to indices and back

Description

A word encoding maps words in a vocabulary to numeric indices.

To encode documents as matrices of word or n-gram counts, use `encode`.

Creation

Syntax

```
enc = wordEncoding(documents)
enc = wordEncoding(words)
enc = wordEncoding(documents,Name,Value)
```

Description

`enc = wordEncoding(documents)` creates a word encoding from the words in `documents`.

`enc = wordEncoding(words)` creates a word encoding from an array of words.

`enc = wordEncoding(documents,Name,Value)` specifies additional options using one or more name-value pair arguments. For example, `'Order','frequency'` assigns lower indices to more frequent words.

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`words` — Input words
string vector | character vector | cell array of character vectors

Input words, specified as a string vector, character vector, or cell array of character vectors. If you specify `words` as a character vector, then the function treats the argument as a single word.

Data Types: `string` | `char` | `cell`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Order','frequency'` sorts the indices by the total frequency in the documents in descending order.

`Order` — Sorting of indices
`'first-seen'` (default) | `'frequency'`

Sorting of indices, specified as the comma-separated pair consisting of `'Order'` and one of the following:

- `'first-seen'` – Assign indices to the words in the order in which they occur in the documents.
- `'frequency'` – Assign indices to the words sorted by total frequency in the documents in descending order.

If `'Order'` is `'frequency'` and multiple words have the same frequency, then the function does not assign indices in any particular order.

`MaxNumWords` — Maximum number of words to encode
`Inf` (default) | positive integer

Maximum number of words to encode, specified as a positive integer or `Inf`. The function first sorts the indices according to the `'Order'` option and then encodes the top `MaxNumWords` words. If `MaxNumWords` is `Inf`, then the function encodes all the words in the input documents.

Properties

`NumWords` — Number of words in model
nonnegative integer

Number of words in the model, specified as a nonnegative integer.

`Vocabulary` — Unique words in model
string vector

Unique words in the model, specified as a string vector.

Data Types: `string`

Object Functions

ind2word	Map encoding index to word
word2ind	Map word to encoding index
isVocabularyWord	Test if word is member of word embedding or encoding

Examples

Create Word Encoding

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
documents(1:10)
```

```
ans = 
  10x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youth
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair th
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial t
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lovs
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt hap
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art belo
```

Create a word encoding.

```
enc = wordEncoding(documents)
```

```
enc = 
  wordEncoding with properties:

      NumWords: 3092
    Vocabulary: [1x3092 string]
```

Create Word Encoding from Word Embedding

To create a word encoding from a word embedding, input the word embedding vocabulary to the `wordEncoding` function as a list of words.

Load pretrained word embedding.

```
emb = fastTextWordEmbedding;
```

Extract the vocabulary.

```
words = emb.Vocabulary;
```

Create a word encoding using the vocabulary.

```
enc = wordEncoding(words)
```

```
enc = 
  wordEncoding with properties:

      NumWords: 999994
    Vocabulary: [1×999994 string]
```

To initialize the corresponding word embedding layer in a deep learning network with the word embedding weights, use the `word2vec` function to extract the layer weights and set the `'Weights'` name-value pair of the `wordEmbeddingLayer` function. The word embedding layer expects columns of word vectors, so you must transpose the output of the `word2vec` function.

```
dimension = emb.Dimension;
numWords = numel(words);
```

```
layer = wordEmbeddingLayer(dimension,numWords, ...
    'Weights',word2vec(emb,words)')
```

```
layer = 
  WordEmbeddingLayer with properties:

         Name: ''

   Hyperparameters
    Dimension: 300
     NumWords: 999994

   Learnable Parameters
      Weights: [300×999994 single]

  Show all properties
```

Create Word Encoding of Top Words in Documents

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
documents(1:10)
```

```
ans = 
  10x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy yout
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair t
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial 
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lov
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt ha
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art bel
```

Create a word encoding. Sort the indices by frequency and encode only the top 100 words.

```
enc = wordEncoding(documents, ...
    'Order','frequency', ...
    'MaxNumWords',100)
```

```
enc = 
  wordEncoding with properties:
      NumWords: 100
    Vocabulary: [1x100 string]
```

View the words corresponding to indices 1, 2, and 3 using the `ind2word` function.

```
idx = [1 2 3];
words = ind2word(enc,idx)
```

```
words = 1x3 string
    "thy"    "thou"    "love"
```

Map Encoding Indices to Words

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
documents(1:10)
```

```
ans = 
  10x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youtl
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair tl
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lov
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt ha
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art bel
```

Create a word encoding.

```
enc = wordEncoding(documents)
```

```
enc = 
  wordEncoding with properties:

      NumWords: 3092
    Vocabulary: [1x3092 string]
```

View the words corresponding to indices 1, 3, and 5 using the `ind2word` function.

```
idx = [1 3 5];
words = ind2word(enc,idx)
words = 1x3 string
    "fairest"    "desire"    "thereby"
```

Map Words to Encoding Indices

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
documents(1:10)
```

```
ans =
  10x1 tokenizedDocument:

    70 tokens: fairest creatures desire increase thereby beautys rose might never die riper time
    71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys field thy youth
    65 tokens: look thy glass tell face thou viewest time face form another whose fresh repair th
    71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys legacy natures
    61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants same unfair
    68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make sweet vial t
    64 tokens: lo orient gracious light lifts up burning head eye doth homage newappearing sight
    70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights joy why lovs
    70 tokens: fear wet widows eye thou consumst thy self single life ah thou issueless shalt hap
    69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt thou art belo
```

Create a word encoding.

```
enc = wordEncoding(documents)
```

```
enc =
  wordEncoding with properties:

      NumWords: 3092
    Vocabulary: [1x3092 string]
```

Map the words "rose", "love", and "beauty" to encoding indices using the `word2ind` function.

```
words = ["rose" "love" "beauty"];
idx = word2ind(enc,words)
```

```
idx = 1×3

     7   387    79
```

Convert Documents to Sequences of Word Indices

Load the factory reports data and create a `tokenizedDocument` array.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
textData = data.Description;
documents = tokenizedDocument(textData);
```

Create a word encoding.

```
enc = wordEncoding(documents);
```

Convert the documents to sequences of word indices.

```
sequences = doc2sequence(enc,documents);
```

View the sizes of the first 10 sequences. Each sequence is a 1-by-*S* vector, where *S* is the number of word indices in the sequence. Because the sequences are padded, *S* is constant.

```
sequences(1:10)
```

```
ans=10×1 cell array
    {[           0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10]}
    {[  0 0 0 0 0 0 11 12 13 14 15 2 16 17 18 19 10]}
    {[    0 0 0 0 0 0 20 2 21 22 7 23 24 25 7 26 10]}
    {[        0 0 0 0 0 0 0 0 0 0 0 27 28 6 7 18 10]}
    {[        0 0 0 0 0 0 0 0 0 0 0 0 29 30 7 31 10]}
    {[    0 0 0 0 0 0 0 32 33 6 7 34 35 36 37 38 10]}
    {[      0 0 0 0 0 0 0 0 0 39 40 36 41 6 7 42 10]}
    {[    0 0 0 0 0 0 0 0 43 44 22 45 46 47 7 48 10]}
    {[      0 0 0 0 0 0 0 0 0 0 49 50 17 7 51 48 10]}
    {[0 0 0 0 52 8 53 36 54 55 56 57 58 59 22 60 10]}
```

See Also

doc2sequence | fastTextWordEmbedding | ind2word | isVocabularyWord | tokenizedDocument | word2ind | word2vec | wordEmbedding | wordEmbeddingLayer

Topics

"Train a Sentiment Classifier"
"Classify Text Data Using Deep Learning"
"Visualize Word Embeddings Using Text Scatter Plots"
"Prepare Text Data for Analysis"

Introduced in R2018b

writeTextDocument

Write documents to text file

Syntax

```
writeTextDocument(documents,filename)
writeTextDocument(documents,filename,'Append',true)
```

Description

`writeTextDocument(documents,filename)` writes `documents` to the specified text file. The function writes one document per line with a space between each word in UTF-8.

`writeTextDocument(documents,filename,'Append',true)` appends to the file instead of overwriting.

Examples

Write Documents to Text File

Write an array of documents to a text file.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"])
```

```
documents = 
  2x1 tokenizedDocument:

    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

```
filename = "documents.txt";
writeTextDocument(documents,filename)
```

Append Documents to Text File

Write an array of documents to a text file by appending the documents one at a time.

Create an array of tokenized documents.

```
documents = tokenizedDocument([
    "an example of a short sentence"
    "a second short sentence"])
```

```
documents = 
  2x1 tokenizedDocument:
    6 tokens: an example of a short sentence
    4 tokens: a second short sentence
```

Write the first document to the file.

```
filename = "documents.txt";
writeTextDocument(documents(1),filename)
```

View the contents of the file using `extractFileText`.

```
str = extractFileText(filename)
```

```
str = 
"an example of a short sentence"
```

Append the second document to the text file.

```
writeTextDocument(documents(2),filename,'Append',true)
```

View the contents of the file using `extractFileText`.

```
str = extractFileText(filename)
```

```
str = 
    "an example of a short sentence
     a second short sentence"
```

Input Arguments

`documents` — Input documents
`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`filename` — Name of file
string scalar | character vector

Name of the file, specified as a string scalar or character vector.

Data Types: `string` | `char`

See Also

`extractFileText` | `extractHTMLText` | `readPDFFormData` | `tokenizedDocument`

Topics
“Extract Text Data from Files”
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

writeWordEmbedding

Write word embedding file

Syntax

```
writeWordEmbedding(emb,filename)
```

Description

`writeWordEmbedding(emb,filename)` writes the word embedding `emb` to the file `filename`. The function writes the vocabulary in UTF-8 in word2vec text format.

Examples

Write Word Embedding to File

Train a word embedding and write it to a text file.

Load the example data. The file `sonnetsPreprocessed.txt` contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from `sonnetsPreprocessed.txt`, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

Train a word embedding using `trainWordEmbedding`.

```
emb = trainWordEmbedding(documents)
```

```
Training: 100% Loss: 2.88661  Remaining time: 0 hours 0 minutes.

emb = 
  wordEmbedding with properties:

     Dimension: 100
    Vocabulary: [1x401 string]
```

Write the word embedding to a text file.

```
filename = "exampleSonnetsEmbedding.vec";
writeWordEmbedding(emb,filename)
```

Read the word embedding file using `readWordEmbedding`.

```
emb = readWordEmbedding(filename)
```

```
emb = 
  wordEmbedding with properties:
     Dimension: 100
    Vocabulary: [1x401 string]
```

Input Arguments

emb — Input word embedding
`wordEmbedding` object

Input word embedding, specified as a `wordEmbedding` object.

`filename` — Name of file
string scalar | character vector

Name of the file, specified as a string scalar or character vector.

Data Types: `string` | `char`

See Also

`doc2sequence` | `fastTextWordEmbedding` | `readWordEmbedding` | `tokenizedDocument` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbedding` | `wordEmbeddingLayer` | `wordEncoding`

Topics

“Train a Sentiment Classifier”
“Classify Text Data Using Deep Learning”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”

Introduced in R2017b