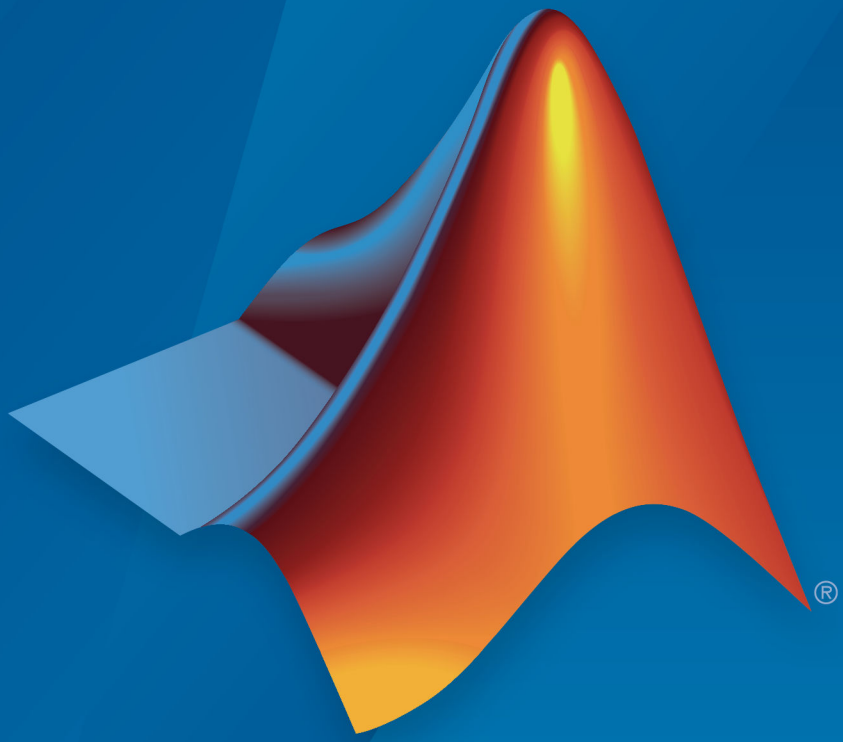
Text Analytics Toolbox™

Reference



MATLAB®

R2018a

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.
3 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)

Contents

Functions — Alphabetical List

abbreviations

Table of common abbreviations

Syntax

```
tbl = abbreviations
```

Description

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

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
    ____________    _______

      "aba"         regular
      "abc"         regular
      "abf"         regular
      "abh"         regular
      "abohm"       regular
      "abs"         regular
      "abt"         regular
      "abv"         regular
```

Output Arguments

`tbl` — Table of abbreviations

table

Table of abbreviations. The `addSentenceDetails` and `splitSentences` functions, by default, use this table to detect sentence boundaries.

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	`fig`	`"See fig. 3."`	`"See fig. 3."`

Usage	Behavior	Example Abbreviation	Example Text	Detected Sentences

	break at the trailing period.		`"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."`

See Also

`addSentenceDetails` | `splitSentences` | `stopWords` | `tokenDetails` | `tokenizedDocument`

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 directory, 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 filenames "`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: {
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                               ... and 1 more
                              }
                 UniformRead: 0
                     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 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.

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”

Introduced in R2017b

addSentenceDetails

Add sentence numbers to documents

Syntax

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

Description

`newDocuments = addSentenceDetails(documents)` detects the sentence boundaries in `documents`. To get the sentence information from `newDocuments`, use `tokenDetails`.

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

Examples

Add Sentence Details to Documents

Create a tokenized document from the text in `exampleSonnet1.txt`.

```
filename = "exampleSonnet1.txt";
str = extractFileText(filename);
document = tokenizedDocument(str)
```

```
document =
  tokenizedDocument:

   124 tokens: From fairest creatures we desire increase , That thereby beauty's rose r
```

View the token details of the first 15 tokens.

```
details = tokenDetails(document);
head(details,15)
```

```
ans=15×4 table
       Token       DocumentNumber    LineNumber       Type
    ___________    ______________    __________    ___________

    "From"              1               1         letters
    "fairest"           1               1         letters
    "creatures"         1               1         letters
    "we"                1               1         letters
    "desire"            1               1         letters
    "increase"          1               1         letters
    ","                 1               1         punctuation
    "That"              1               2         letters
    "thereby"           1               2         letters
    "beauty's"          1               2         other
    "rose"              1               2         letters
    "might"             1               2         letters
    "never"             1               2         letters
    "die"               1               2         letters
    ","                 1               2         punctuation
```

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 15 tokens.

```
document = addSentenceDetails(document);
details = tokenDetails(document);
head(details,15)
```

```
ans=15×5 table
       Token       DocumentNumber    SentenceNumber    LineNumber       Type
    ___________    ______________    ______________    __________    ___________

    "From"              1                 1               1         letters
    "fairest"           1                 1               1         letters
    "creatures"         1                 1               1         letters
    "we"                1                 1               1         letters
    "desire"            1                 1               1         letters
    "increase"          1                 1               1         letters
    ","                 1                 1               1         punctuation
    "That"              1                 1               2         letters
    "thereby"           1                 1               2         letters
```

```
    "beauty's"          1                1                2          other
    "rose"              1                1                2          letters
    "might"             1                1                2          letters
    "never"             1                1                2          letters
    "die"               1                1                2          letters
    ","                 1                1                2          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 single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Abbreviations',["cm" "mm" "in"]` species 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."`

Usage	Behavior	Example Abbreviation	Example Text	Detected Sentences
	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.

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`

Output Arguments

`newDocuments` — Updated documents
`tokenizedDocument` array

Updated documents, returned as a `tokenizedDocument` array. To get the sentence information from `newDocuments`, use `tokenDetails`.

See Also

`abbreviations` | `splitSentences` | `stopWords` | `tokenDetails` | `tokenizedDocument`

Introduced in R2018a

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 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.

`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:

 (1,1) 70 tokens: fairest creatures desire increase thereby beautys rose might never di
 (2,1) 71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys fi
 (3,1) 65 tokens: look thy glass tell face thou viewest time face form another whose fr
 (4,1) 71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys le
 (5,1) 61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants
 (6,1) 68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make
 (7,1) 64 tokens: lo orient gracious light lifts up burning head eye doth homage newapp
 (8,1) 70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights
 (9,1) 70 tokens: fear wet widows eye thou consumst thy self single life ah thou issue
(10,1) 69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt
```

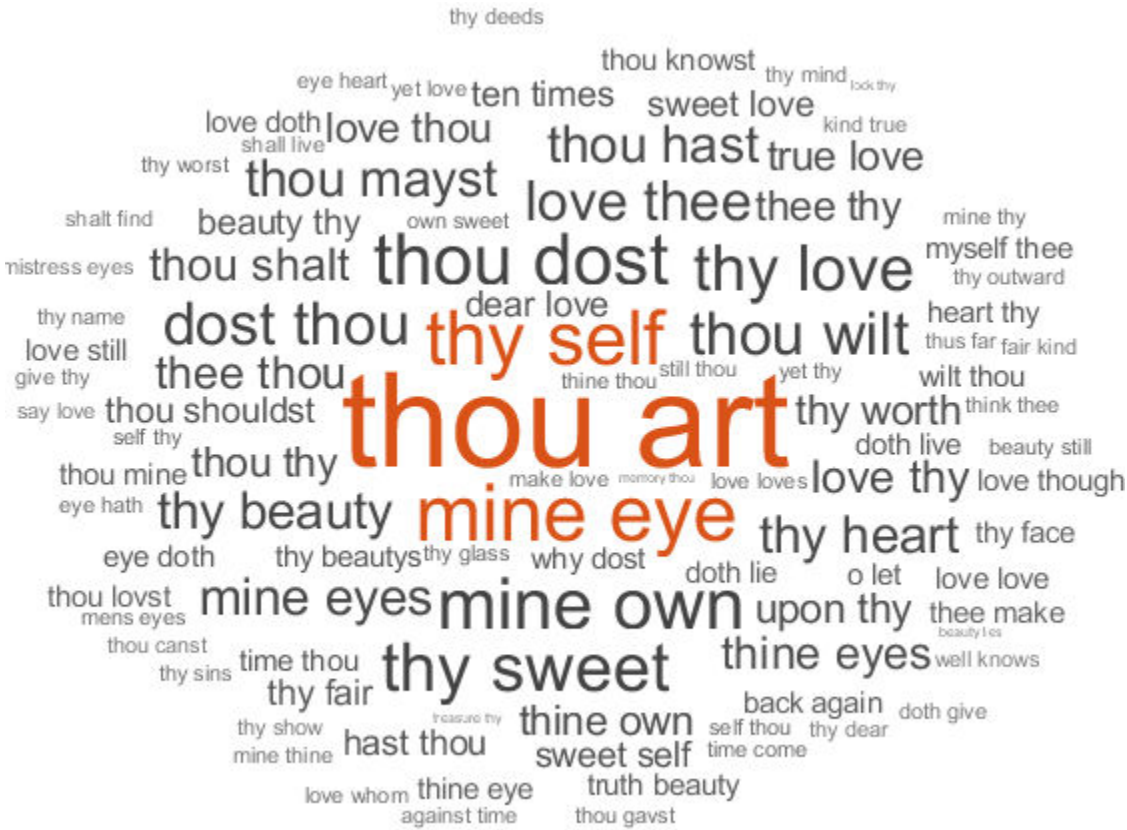
Create a bag-of-n-grams mdoel.

```
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 array
    "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
```

- “Analyze Text Data Using Multiword Phrases”
- “Analyze Text Data Using Topic Models”
- “Visualize Text Data Using Word Clouds”
- “Classify Text Data Using Deep Learning”

See Also

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

Topics

“Analyze Text Data Using Multiword Phrases”
“Analyze Text Data Using Topic Models”
“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 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.

`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.

`Vocabulary` — Unique words in model
string vector

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

Data Types: `string`

`NumWords` — Number of words seen
nonnegative integer

Number of words 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
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 directory, 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 filenames "`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: {
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                               ... and 1 more
                              }
                 UniformRead: 0
                     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    2.4
         0         0         0         0         0    4.5287         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0
         0         0         0         0         0    2.2644         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    2.2644         0         0
         0         0    2.7344         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`.

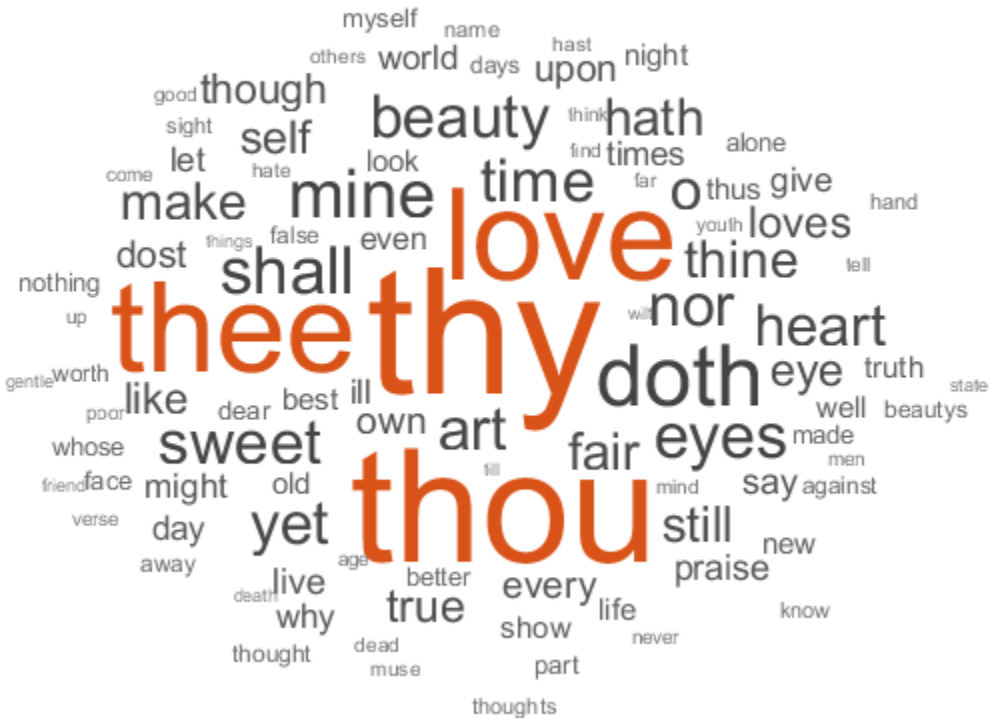
```
bag = bagOfWords(documents)
```

```
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 directory, 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 filenames "`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 = 5x1 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
```

```
Starting parallel pool (parpool) using the 'local' profile ...
connected to 12 workers.
```

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

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

```
bag =
  bagOfWords with properties:

          Counts: [5x3275 double]
      Vocabulary: [1x3275 string]
        NumWords: 3275
    NumDocuments: 5
```

- “Prepare Text Data for Analysis”
- “Analyze Text Data Using Topic Models”
- “Analyze Text Data Using Multiword Phrases”
- “Visualize Text Data Using Word Clouds”
- “Classify Text Data Using Deep Learning”

Tips

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

See Also

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

Topics

“Prepare Text Data for Analysis”
“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

context

Search documents for word occurrences in context

Syntax

```
T = context(documents,word)
T = context(documents,word,contextLength)
T = context( ___ ,'Source',source)
```

Description

`T = context(documents,word)` searches for occurrences of `word` in `documents` and returns a table showing `word` in context and its locations.

`T = context(documents,word,contextLength)` specifies the length of the context to return.

`T = context( ___ ,'Source',source)` displays the context in the original source string `source` if the word is found.

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 array
    "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"
```

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 array
    "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 afterwards.

```
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 =

  23x3 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
    "hou art assur mine ; and life no longer than thy love "         92         20
    " in the least of them my life hath end . i see a bette"         92         56
    "onst mind , sinc that my life on thy revolt doth lie ."         92         89
    "me faster than time wast life , so thou prevent'st hi "        100        118
    "at did not better for my life provid than public mean "        111         26
    "she threw , and sav'd my life , sai ' not you ' .     "        145        113
    "i nymph that vow'd chast life to keep came trip by ; b"        154         22
```

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 =

  23x3 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
"ou art assured mine; And life no longer than thy love "       92        20
" in the least of them my life hath end. I see a better"       92        56
"tant mind, Since that my life on thy revolt doth lie. "       92        89
" faster than Time wastes life, So thou prevent'st his "      100       118
"at did not better for my life provide Than public mean"      111        26
" she threw, And sav'd my life, saying 'not you'.      "      145       113
"nymphs that vow'd chaste life to keep Came tripping by"      154        22
```

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`

`contextLength` — Context length

25 (default) | positive integer

Context length, specified as a positive integer.

`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`.

Output Arguments

`T` — Table of contexts
table

Table of contexts with these columns:

`Context`	String containing the queried word in context
`Document`	Numeric index of the document containing the word
`Word`	Numeric index of the word 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

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 array
    "<>"    "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` | `tokenizedDocument`

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."];
str = erasePunctuation(str)
```

```
str = 2x1 string array
    "An example of a short sentence"
    "A second short sentence"
```

```
str = lower(str)
```

```
str = 2x1 string array
    "an example of a short sentence"
    "a second short sentence"
```

```
documents = tokenizedDocument(str)

documents =
  2x1 tokenizedDocument:

(1,1) 6 tokens: an example of a short sentence
(2,1) 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:

(1,1) 6 tokens: an example of a short sentence
(1,2) 4 tokens: a second short sentence
```

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

```
C = 1x2 cell array
    {1x6 string}    {1x4 string}
```

View the first element of the cell array.

```
C{1}
```

```
ans = 1x6 string array
    "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

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`.

`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:

(1,1) 6 tokens: an example of a short sentence
(2,1) 4 tokens: a second short sentence


func = @reverse;
newDocuments = docfun(func,documents)

newDocuments =
  2x1 tokenizedDocument:

(1,1) 6 tokens: na elpmaxe fo a trohs ecnetnes
(2,1) 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:
```

```
(1,1) 6 tokens: an example of a short sentence
(2,1) 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:

(1,1) 6 tokens: _det _noun _prep _det _adj _noun
(2,1) 4 tokens: _det _adj _adj _noun


func = @plus;
newDocuments = docfun(func,documents1,documents2)

newDocuments =
  2x1 tokenizedDocument:

(1,1) 6 tokens: an_det example_noun of_prep a_det short_adj sentence_noun
(2,1) 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:

(1,1) 12 tokens: an example of a short sentence _det _noun _prep _det _adj _noun
(2,1)  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`.
- `[newWords1,...,newWordsN] = 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

`bagOfWords` | `lower` | `normalizeWords` | `regexprep` | `replace` | `tokenDetails` | `tokenizedDocument` | `upper`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

encode

Encode documents as matrix of word or n-gram counts

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 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”.

Syntax

```
counts = encode(bag,documents)
counts = encode(bag,words)
counts = encode( ___ ,Name,Value)
```

Description

`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:

(1,1) 3 tokens: a new sentence
(2,1) 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 array
    "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:

(1,1) 3 tokens: a new sentence
(2,1) 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.

`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 single 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` | `topkngrams` | `topkwords`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

erasePunctuation

Erase punctuation from text and documents

Syntax

```
newStr = erasePunctuation(str)
newDocuments = erasePunctuation(documents)
```

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.

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, 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."
    "A short, small, and simple sentence"])

documents =
  2x1 tokenizedDocument:

(1,1) 7 tokens: An example of a short sentence .
(2,1) 8 tokens: A short , small , and simple sentence


newDocuments = erasePunctuation(documents)

newDocuments =
  2x1 tokenizedDocument:

(1,1) 6 tokens: An example of a short sentence
(2,1) 6 tokens: A short small and simple sentence
```

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.

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.

Definitions

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	”

Category	Category Code	Number of Characters	Example Character
Punctuation, Initial quote	[Pi]	12	“
Punctuation, Other	[Po]	566	!
Punctuation, Open	[Ps]	75	(
Symbol, Currency	[Sc]	54	$
Symbol, Modifier	[Sk]	121	^
Symbol, Math	[Sm]	948	+
Symbol, Other	[So]	5855	¦

For more information, see [1].

Tips

- `erasePunctuation` removes punctuation characters from URLs and HTML tags. This may prevent 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`.

References

[1] *Unicode Character Categories*. http://www.fileformat.info/info/unicode/category/index.htm

See Also

`decodeHTMLEntities` | `eraseTags` | `eraseURLs` | `tokenizedDocument`

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` | `tokenizedDocument`

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 = "See http://mathworks.com for more information.";
newStr = eraseURLs(str)
```

```
newStr =
"See  for more information."
```

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` | `tokenizedDocument` | `topLevelDomains`

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 directory, 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 filenames "`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: {
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                              ' ...\ib0BF173\5\tp3d708a43\textanalytics-ex73762432\exar
                               ... and 1 more
                              }
                 UniformRead: 0
                     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 single 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

`'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-847'`
`'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. 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.

`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. Only has an effect 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. Only has an effect if the input file is a PDF. The function, by default, reads all pages from PDF.

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`.

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(code,'ExtractionMethod',ex)
```

Description

`str = extractHTMLText(code)` parses the HTML code in `code` and extracts the article text.

`str = extractHTMLText(code,'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 `webread` to read the HTML code.

```
url = "https://www.mathworks.com/help/textanalytics";
code = webread(url);
str = extractHTMLText(code)
```

```
str =
    'Text Analytics Toolbox™ provides algorithms and visualizations for preprocessing,

     Text Analytics Toolbox includes tools for processing raw text from sources such as

     Using machine learning techniques such as LSA, LDA, and word embeddings, you can t
```

Input Arguments

code — HTML code

string scalar | character vector | scalar cell array containing a character vector

HTML code, specified as a string scalar, a character vector, or a scalar cell array containing a character vector.

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`

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` | `readPDFFormData` | `tokenizedDocument` | `webread` | `writeTextDocument`

Topics

“Extract Text Data from Files”
“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2018a

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.

Output Arguments

`emb` — Pretrained word embedding

`wordEmbedding` object

Pretrained word embedding, returned as a `wordEmbedding` object.

See Also

`ismember` | `readWordEmbedding` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbedding` | `writeWordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2018a

fitlda

Fit latent Dirichlet allocation (LDA) model

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.

Syntax

```
mdl = fitlda(bag,numTopics)
mdl = fitlda(counts,numTopics)
mdl = fitlda(___,Name,Value)
```

Description

`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.

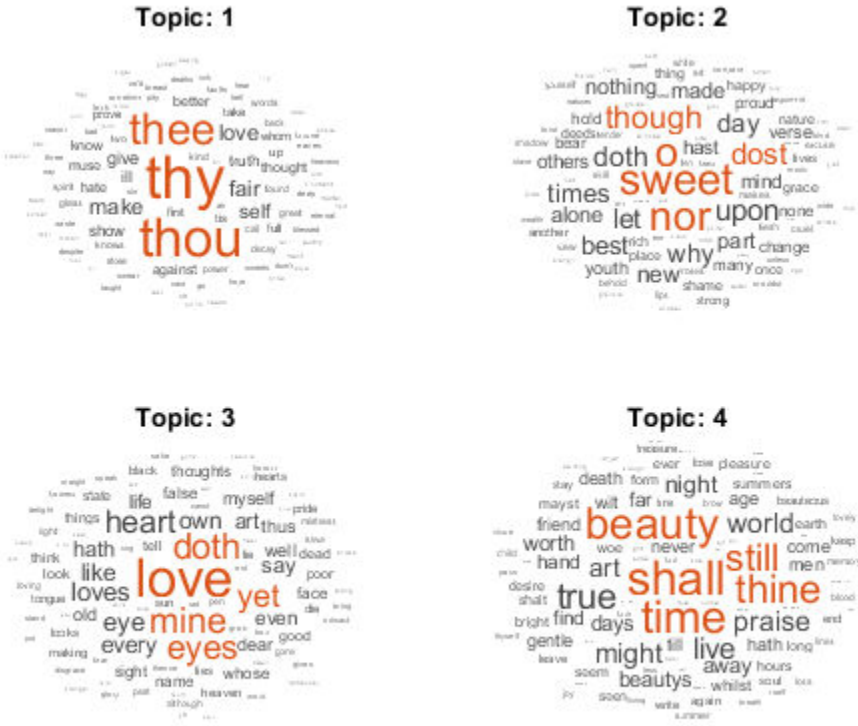
```
numTopics = 4;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0772073 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 |       0.08 | 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 |       0.08 | 9.4193e-04 |  1.111e+03 |         1.000 |             0 |
|          5 |       0.03 | 3.7079e-04 |  1.108e+03 |         1.000 |             0 |
|          6 |       0.03 | 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]
                        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 a 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
```

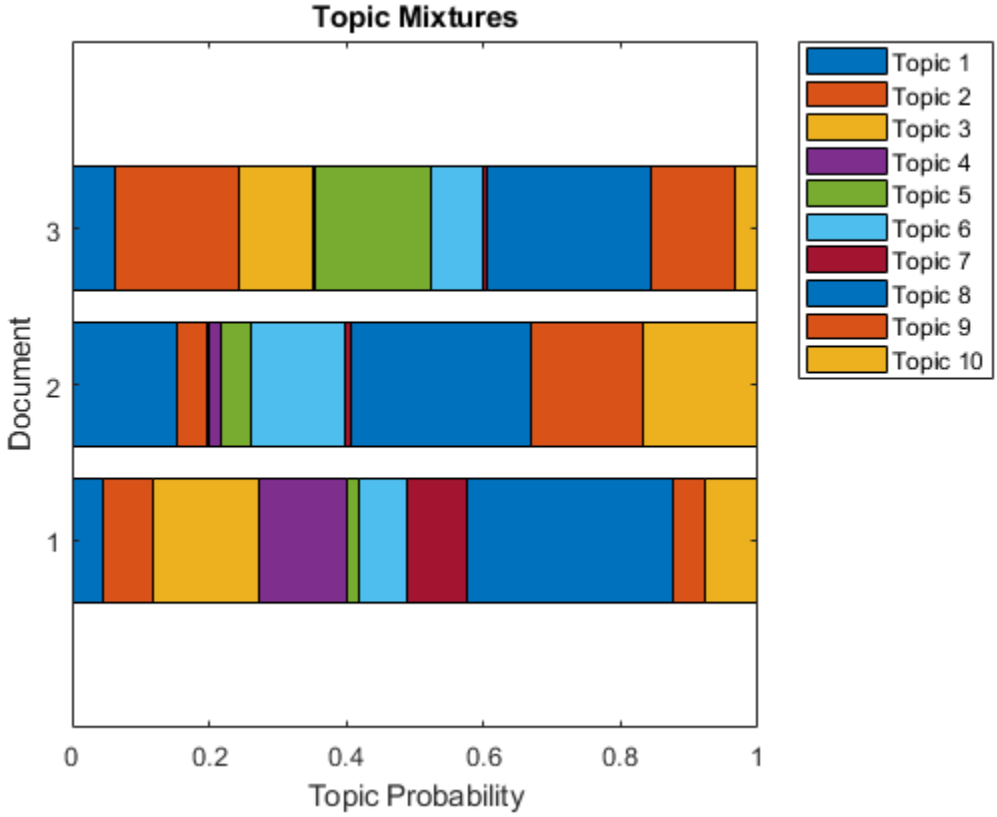
Fit an LDA model with 10 topics. To disable the verbose output, set `'Verbose'` to 0.

```
numTopics = 10;
mdl = fitlda(counts,numTopics,'Verbose',0)
```

```
mdl =
  ldaModel with properties:

                     NumTopics: 10
            WordConcentration: 1
           TopicConcentration: 2.5000
      CorpusTopicProbabilities: [1x10 double]
    DocumentTopicProbabilities: [154x10 double]
        TopicWordProbabilities: [3092x10 double]
                    Vocabulary: [1x3092 string]
                       FitInfo: [1x1 struct]
```

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 "+(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 0.0476809 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 |       0.07 | 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]
       FitInfo: [1x1 struct]
```

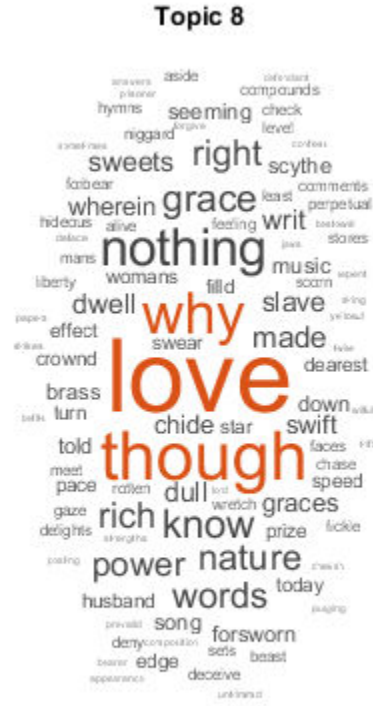
Predict the top topics for an array of new documents.

```
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 the n-grams as individual words.

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 single 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 $\alpha = \alpha_0 \begin{pmatrix} p_1 & p_2 & \cdots & p_K \end{pmatrix}$ 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 $\alpha = \alpha_0 \begin{pmatrix} p_1 & p_2 & \cdots & p_K \end{pmatrix}$ 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 1-96.

Example: `'InitialTopicConcentration',25`

`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 1-96.

`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 1-99.

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.

`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.

Definitions

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 *v*th word of the vocabulary appearing in the *i*th 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.

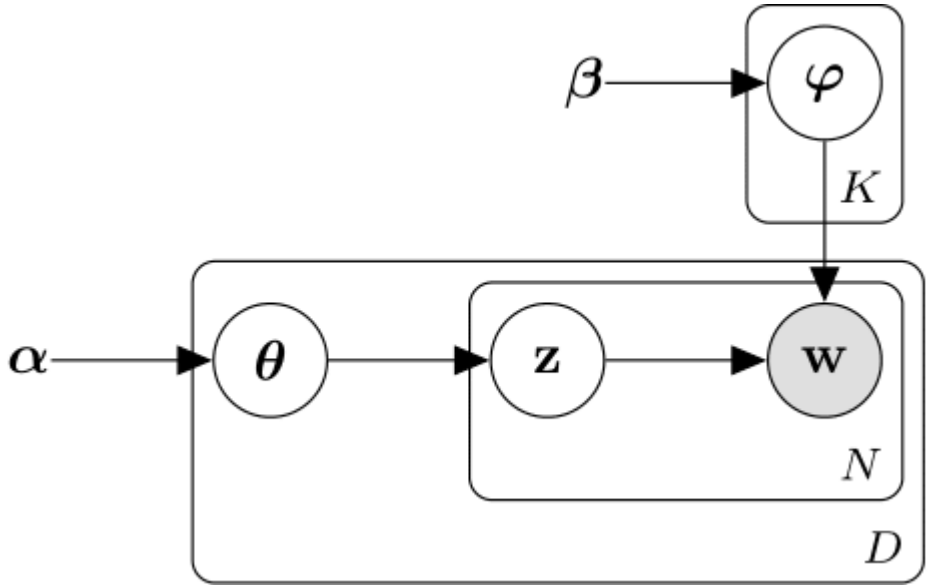
Under this generative process, the joint distribution of a document with words $w_1,\ldots,w_N$, with topic mixture θ, and with topic indices $z_1,\ldots,z_N$ is given by

$$p(\theta, z, w \mid \alpha, \varphi) = p(\theta \mid \alpha)\prod_{n=1}^{N} p(z_n \mid \theta)\, p(w_n \mid z_n, \varphi),$$

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*:

$$p(w|\alpha,\varphi)=\int_{\theta}p(\theta|\alpha)\prod_{n=1}^{N}\sum_{z_n}p(z_n|\theta)\,p(w_n|z_n,\varphi)d\theta.$$

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

$$p(\theta\#\,\alpha)=\frac{1}{B(\alpha)}\prod_{i=1}^{K}\theta_i^{\alpha_i-1},$$

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.

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

`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    0.1
    0.2547    0.5576   -0.0095    0.5660   -0.0643   -0.1236   -0.0082    0.0522   -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 the n-grams as individual words.

`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 single 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”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

ismember

Test word is member of word embedding

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.

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]
```

Test if the words `"I"`, `"love"`, and `"MATLAB"` are in the word embedding.

```
words = ["I" "love" "MATLAB"]
```

```
words = 1x3 string array
    "I"    "love"    "MATLAB"


tf = ismember(emb,words)

tf = 1x3 logical array

   0   1   0
```

Input Arguments

emb — Input word embedding

word embedding

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`

See Also

`readWordEmbedding` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbedding` | `writeWordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

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 =
  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 directory, 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 filenames "`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 = 5x1 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
```

```
Starting parallel pool (parpool) using the 'local' profile ...
connected to 12 workers.
```

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

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

```
bag =
  bagOfWords with properties:

          Counts: [5x3275 double]
      Vocabulary: [1x3275 string]
        NumWords: 3275
    NumDocuments: 5
```

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

`bagOfNgrams` | `bagOfWords`

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:

(1,1) 6 tokens: an example of a short sentence
(2,1) 4 tokens: a second short sentence
```

```
str = joinWords(documents)
```

```
str = 2x1 string array
    "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 array
    "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

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 1-130.

`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 1-130.

`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 *k*th 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`.

`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.

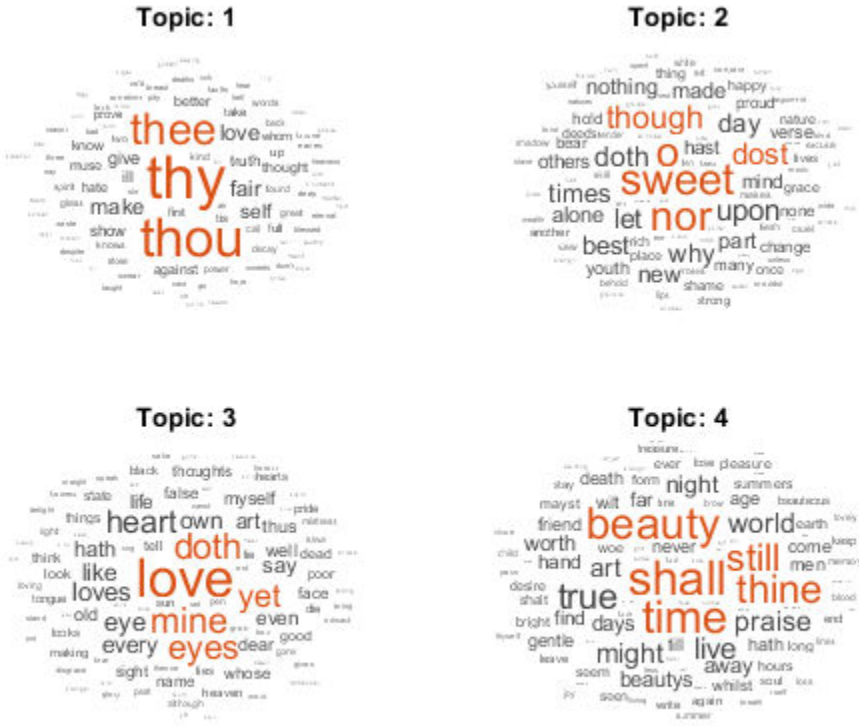
```
numTopics = 4;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0772073 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 |       0.08 | 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 |       0.08 | 9.4193e-04 |  1.111e+03 |         1.000 |             0 |
|          5 |       0.03 | 3.7079e-04 |  1.108e+03 |         1.000 |             0 |
|          6 |       0.03 | 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]
                       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.

```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0747008 seconds.
=================================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=================================================================================================
|          0 |       0.09 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.07 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.08 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.07 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.08 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.08 | 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]
                       FitInfo: [1x1 struct]
```

Find the top 20 words of the first topic.

```
k = 20;
topicIdx = 1;
T = topkwords(mdl,k,topicIdx)
```

```
T=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.

```
T = topkwords(mdl,k,topicIdx,'Scaling','inversemean')
```

```
T=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(T.Word,T.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.

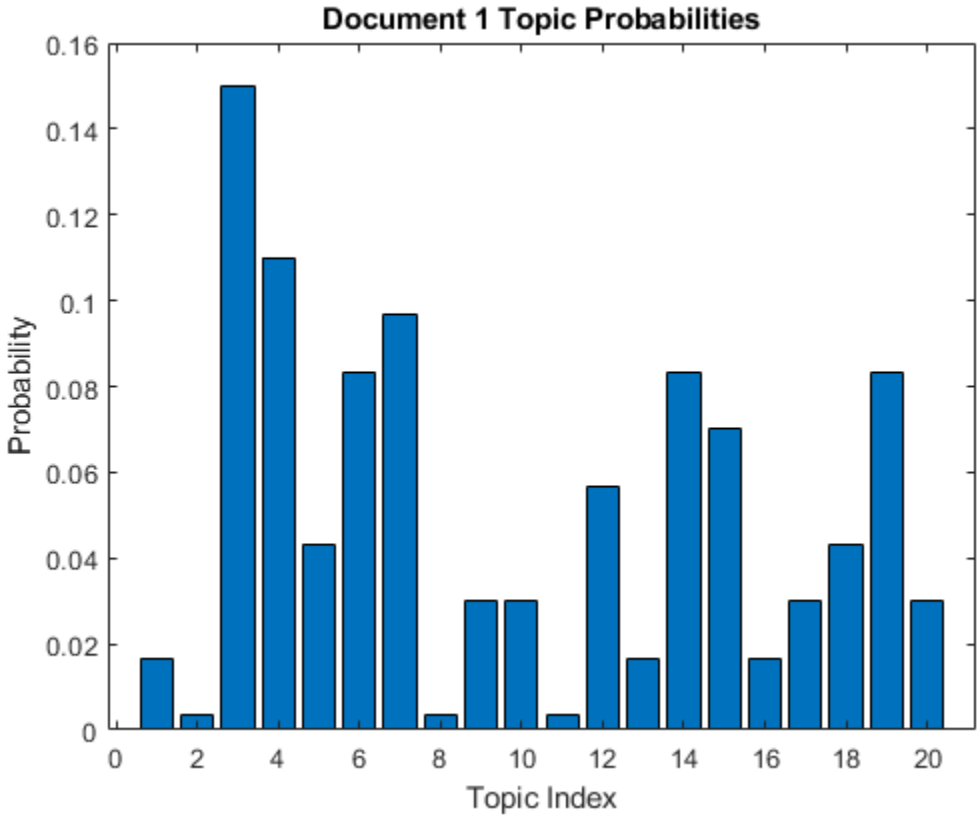
```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0773174 seconds.
=================================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=================================================================================================
|          0 |       0.03 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.10 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.09 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.11 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.10 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.12 | 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]
                     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 0.0476809 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 |       0.07 | 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]
       FitInfo: [1x1 struct]
```

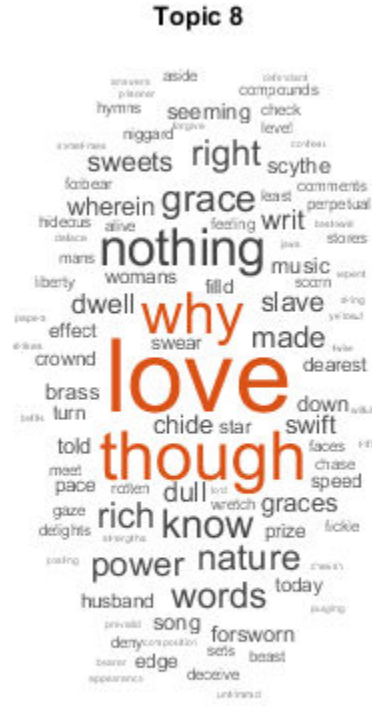
Predict the top topics for an array of new documents.

```
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))
```



- “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”

Definitions

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.

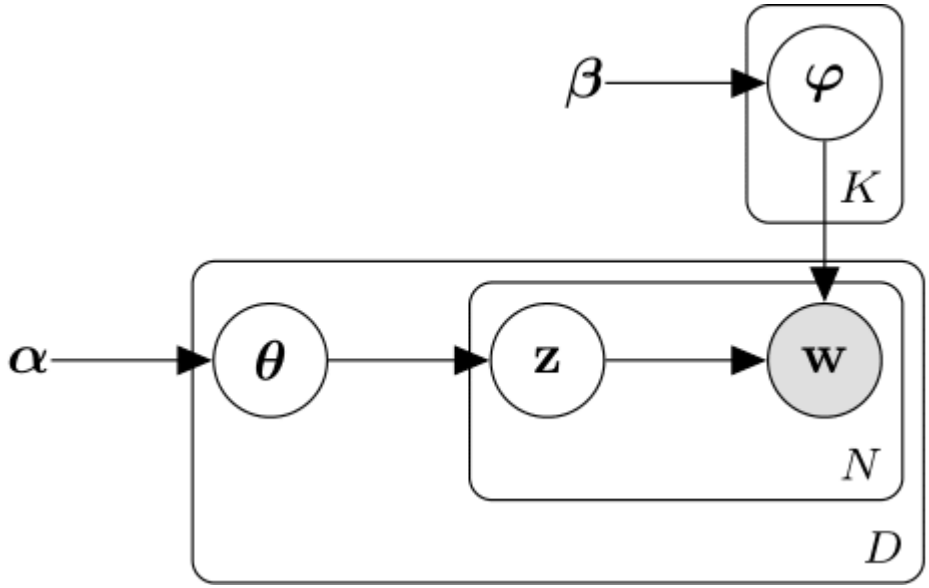
Under this generative process, the joint distribution of a document with words $w_1, \ldots, w_N$, with topic mixture θ, and with topic indices $z_1, \ldots, z_N$ is given by

$$p(\theta, z, w \mid \alpha, \varphi) = p(\theta \mid \alpha) \prod_{n=1}^{N} p(z_n \mid \theta)\, p(w_n \mid z_n, \varphi),$$

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:

$$p(w \mid \alpha, \varphi) = \int_{\theta} p(\theta \mid \alpha) \prod_{n=1}^{N} \sum_{z_n} p(z_n \mid \theta)\, p(w_n \mid z_n, \varphi)\, d\theta.$$

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

$$p(\theta \# \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} \theta_i^{\alpha_i - 1},$$

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

lsaModel

Latent semantic analysis (LSA) model

Description

A latent semantic analysis (LSA) model discovers relationships between documents and the words that they contain. 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    0.1
    0.2547    0.5576   -0.0095    0.5660   -0.0643   -0.1236   -0.0082    0.0522   -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
```

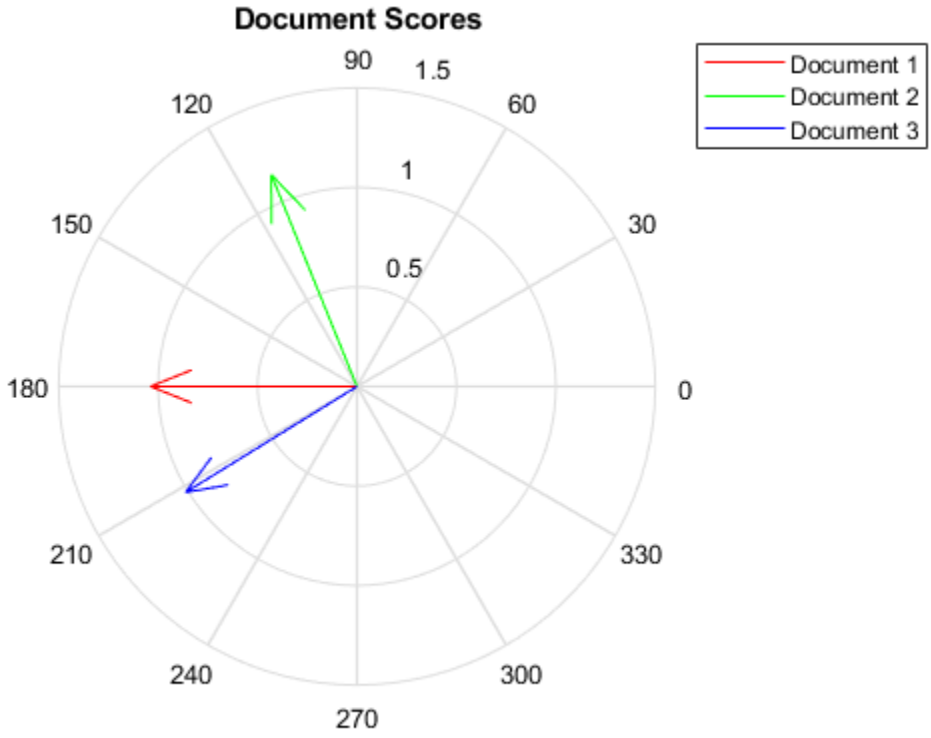
Calculate the cosine distance between the documents score vectors using `pdist`. View the distances in a matrix `D` using `squareform`. `D(i,j)` denotes the distance between document `i` and `j`.

```
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')
```



- “Analyze Text Data Using Topic Models”
- “Prepare Text Data for Analysis”
- “Extract Text Data from Files”

See Also

`bagOfWords` | `ldaModel` | `lsaModel`

Topics

“Analyze Text Data Using Topic Models”

“Prepare Text Data for Analysis”
“Extract Text Data from Files”

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.

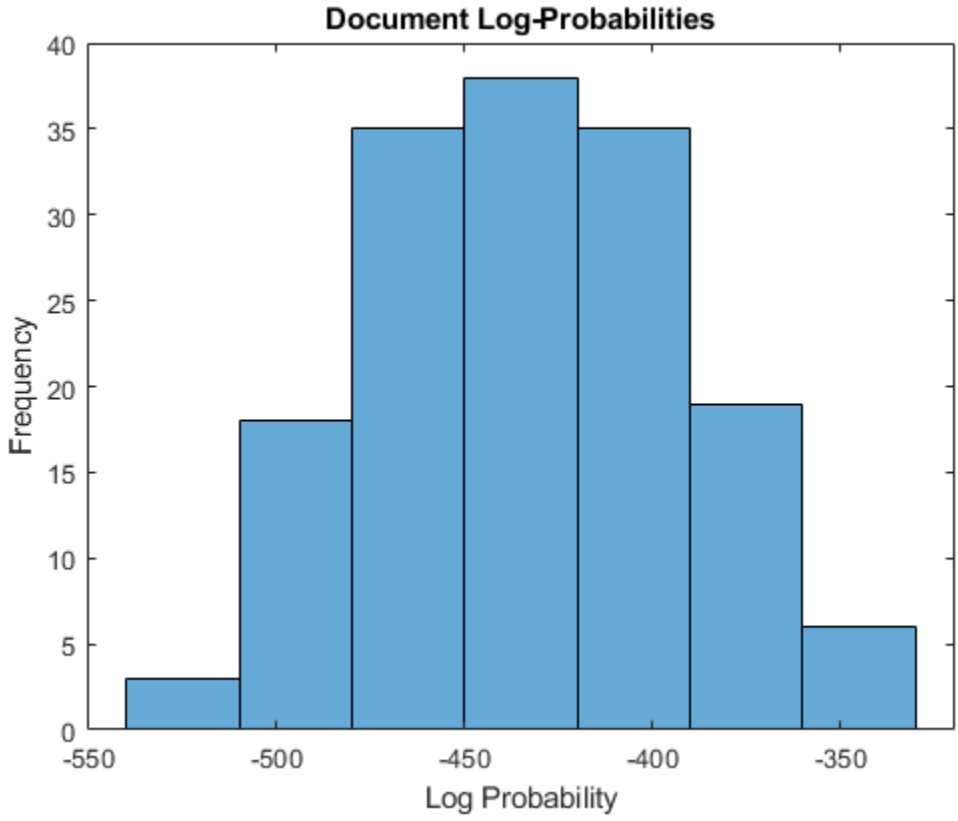
```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.132474 seconds.
=================================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=================================================================================================
|          0 |       0.14 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.07 | 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 |       0.07 | 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]
                      FitInfo: [1x1 struct]
```

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:

(1,1) 76 tokens: poor soul centre sinful earth sinful earth rebel powers array why dost
(2,1) 76 tokens: devouring time blunt thou lions paws make earth devour own sweet brooc
(3,1) 73 tokens: brass nor stone nor earth nor boundless sea sad mortality oersways pow
```

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 0.0673149 seconds.
=================================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=================================================================================================
|          0 |       0.03 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.08 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.11 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.07 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.08 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.07 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.07 | 6.4180e-05 |  7.291e+02 |         5.000 |             0 |
=================================================================================================

mdl =
  ldaModel with properties:
```

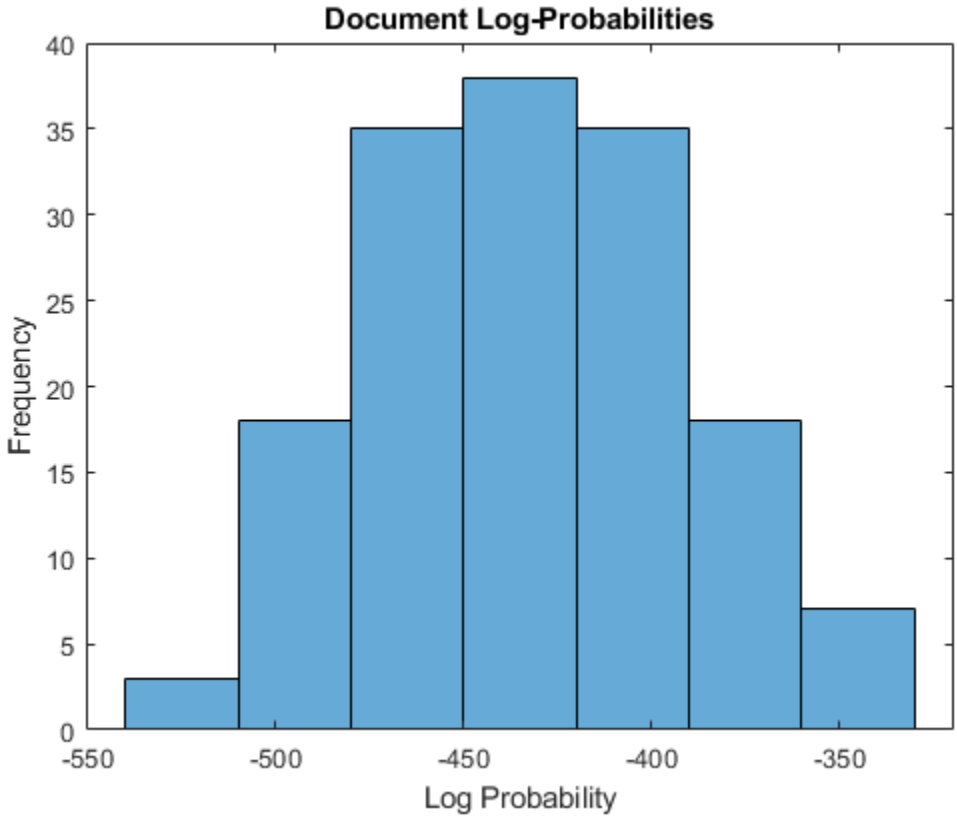
```
                    NumTopics: 20
           WordConcentration: 1
          TopicConcentration: 5
    CorpusTopicProbabilities: [1x20 double]
  DocumentTopicProbabilities: [154x20 double]
      TopicWordProbabilities: [3092x20 double]
                  Vocabulary: [1x3092 string]
                     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.

```
numTopics = 20;
mdl1 = fitlda(bag,numTopics);
```

```
Initial topic assignments sampled in 0.108331 seconds.
=================================================================================
```

```
| Iteration  |  Time per  |  Relative  |  Training  |     Topic      |     Topic      |
|            | iteration  | change in  | perplexity | concentration  | concentration  |
|            | (seconds)  |   log(L)   |            |                |   iterations   |
==========================================================================================
|          0 |       0.05 |            |  1.137e+03 |          5.000 |              0 |
|          1 |       0.10 | 5.5571e-02 |  7.850e+02 |          5.000 |              0 |
|          2 |       0.09 | 5.2541e-03 |  7.582e+02 |          5.000 |              0 |
|          3 |       0.07 | 3.9933e-03 |  7.384e+02 |          5.000 |              0 |
|          4 |       0.08 | 7.2248e-04 |  7.349e+02 |          5.000 |              0 |
|          5 |       0.08 | 3.8814e-03 |  7.164e+02 |          5.000 |              0 |
|          6 |       0.15 | 2.1087e-03 |  7.065e+02 |          5.000 |              0 |
|          7 |       0.08 | 2.0816e-03 |  6.970e+02 |          5.000 |              0 |
|          8 |       0.09 | 1.9510e-03 |  6.882e+02 |          5.000 |              0 |
|          9 |       0.08 | 1.4162e-03 |  6.818e+02 |          5.000 |              0 |
|         10 |       0.07 | 1.4113e-03 |  6.756e+02 |          5.000 |              0 |
|         11 |       0.12 | 1.0378e-03 |  6.710e+02 |         13.255 |             23 |
|         12 |       0.09 | 1.1699e-02 |  7.248e+02 |         16.536 |             20 |
|         13 |       0.07 | 2.5441e-03 |  7.371e+02 |         17.269 |             13 |
|         14 |       0.10 | 9.7803e-04 |  7.418e+02 |         18.155 |             14 |
|         15 |       0.08 | 9.8659e-04 |  7.467e+02 |         18.650 |             11 |
|         16 |       0.07 | 1.9437e-03 |  7.564e+02 |         19.555 |             14 |
|         17 |       0.07 | 1.0115e-03 |  7.513e+02 |         18.746 |             13 |
|         18 |       0.08 | 1.3317e-03 |  7.580e+02 |         19.781 |             15 |
|         19 |       0.07 | 3.1099e-04 |  7.596e+02 |         19.851 |              5 |
|         20 |       0.07 | 8.9869e-04 |  7.550e+02 |         19.181 |             12 |
==========================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic      |     Topic      |
|            | iteration  | change in  | perplexity | concentration  | concentration  |
|            | (seconds)  |   log(L)   |            |                |   iterations   |
==========================================================================================
|         21 |       0.07 | 5.1822e-04 |  7.525e+02 |         19.364 |              7 |
|         22 |       0.10 | 4.5942e-04 |  7.502e+02 |         19.278 |              5 |
|         23 |       0.11 | 1.0923e-04 |  7.496e+02 |         19.250 |              4 |
|         24 |       0.08 | 4.2040e-04 |  7.476e+02 |         19.079 |              6 |
|         25 |       0.07 | 8.8342e-04 |  7.432e+02 |         18.516 |             11 |
|         26 |       0.08 | 9.9164e-05 |  7.427e+02 |         18.606 |              5 |
==========================================================================================
```

Compute the perplexity of the held-out test set.

```
[~,ppl1] = logp(mdl1,documentsTest)
```

```
ppl1 = 781.5692
```

Fit an LDA model with 40 topics to the bag-of-words model.

```
numTopics = 40;
mdl2 = fitlda(bag,numTopics);
```

```
Initial topic assignments sampled in 0.0469017 seconds.
=========================================================================================
| Iteration |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|           | iteration  | change in  | perplexity | concentration | concentration |
|           | (seconds)  |   log(L)   |            |               |   iterations  |
=========================================================================================
|         0 |       0.02 |            |  1.068e+03 |        10.000 |             0 |
|         1 |       0.11 | 8.7406e-02 |  6.095e+02 |        10.000 |             0 |
|         2 |       0.10 | 7.1007e-03 |  5.826e+02 |        10.000 |             0 |
|         3 |       0.10 | 4.1709e-03 |  5.674e+02 |        10.000 |             0 |
|         4 |       0.10 | 4.7632e-03 |  5.506e+02 |        10.000 |             0 |
|         5 |       0.09 | 2.5930e-03 |  5.417e+02 |        10.000 |             0 |
|         6 |       0.10 | 2.8340e-03 |  5.321e+02 |        10.000 |             0 |
|         7 |       0.10 | 4.4464e-04 |  5.336e+02 |        10.000 |             0 |
|         8 |       0.09 | 1.3042e-03 |  5.293e+02 |        10.000 |             0 |
|         9 |       0.10 | 3.2357e-03 |  5.187e+02 |        10.000 |             0 |
|        10 |       0.11 | 2.4565e-04 |  5.179e+02 |        10.000 |             0 |
|        11 |       0.12 | 2.9851e-03 |  5.083e+02 |        16.621 |            17 |
|        12 |       0.10 | 8.3119e-03 |  5.356e+02 |        18.276 |            13 |
|        13 |       0.10 | 4.3005e-03 |  5.503e+02 |        19.955 |            13 |
|        14 |       0.11 | 1.2658e-03 |  5.547e+02 |        20.172 |             6 |
|        15 |       0.15 | 1.1721e-03 |  5.589e+02 |        20.424 |             7 |
|        16 |       0.12 | 9.2352e-04 |  5.621e+02 |        20.961 |             9 |
|        17 |       0.11 | 1.9529e-03 |  5.691e+02 |        21.906 |            11 |
|        18 |       0.11 | 9.4417e-04 |  5.726e+02 |        22.304 |             8 |
|        19 |       0.14 | 3.6839e-04 |  5.739e+02 |        22.569 |             7 |
|        20 |       0.12 | 3.7166e-04 |  5.725e+02 |        22.342 |             6 |
=========================================================================================
| Iteration |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|           | iteration  | change in  | perplexity | concentration | concentration |
|           | (seconds)  |   log(L)   |            |               |   iterations  |
=========================================================================================
|        21 |       0.13 | 5.8456e-04 |  5.704e+02 |        22.095 |             6 |
|        22 |       0.11 | 8.8278e-04 |  5.672e+02 |        21.742 |             7 |
|        23 |       0.12 | 4.1349e-04 |  5.687e+02 |        22.145 |             8 |
|        24 |       0.11 | 2.4727e-03 |  5.599e+02 |        21.493 |             9 |
|        25 |       0.10 | 6.1150e-04 |  5.577e+02 |        21.321 |             5 |
|        26 |       0.10 | 4.3317e-04 |  5.593e+02 |        21.625 |             7 |
|        27 |       0.11 | 3.9528e-04 |  5.607e+02 |        21.798 |             6 |
|        28 |       0.12 | 5.4531e-04 |  5.626e+02 |        21.823 |             4 |
|        29 |       0.11 | 2.5229e-04 |  5.635e+02 |        21.842 |             4 |
|        30 |       0.10 | 9.1549e-04 |  5.668e+02 |        21.834 |             4 |
```

```
|         31 |       0.11 | 1.9169e-03 |  5.600e+02 |        21.630 |             5 |
|         32 |       0.11 | 1.8207e-04 |  5.593e+02 |        21.565 |             4 |
|         33 |       0.11 | 9.4024e-04 |  5.627e+02 |        21.601 |             4 |
|         34 |       0.10 | 6.0374e-04 |  5.605e+02 |        21.538 |             4 |
|         35 |       0.12 | 5.7683e-04 |  5.626e+02 |        21.898 |             8 |
|         36 |       0.11 | 1.0780e-03 |  5.587e+02 |        21.485 |             8 |
|         37 |       0.12 | 3.6849e-05 |  5.589e+02 |        21.460 |             4 |
=========================================================================================
```

Compute the perplexity of the held-out test set.

```
[~,ppl2] = logp(mdl2,documentsTest)
```

```
ppl2 = 808.6732
```

The perplexity is lower for the first model which suggests that this 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 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.

`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 the n-grams as individual words.

`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 single 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:

(1,1) 6 tokens: An Example of a Short Sentence
(2,1) 4 tokens: A Second Short Sentence
```

```
newDocuments = lower(documents)
```

```
newDocuments =
  2x1 tokenizedDocument:
```

```
(1,1) 6 tokens: an example of a short sentence
(2,1) 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

`bagOfWords` | `docfun` | `normalizeWords` | `regexprep` | `replace` | `tokenizedDocument` | `upper`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

normalizeWords

Reduce words to common stems using the Porter stemmer

`normalizeWords` uses the Porter stemmer to group different forms of English words by reducing them to a common stem. This common stem is not necessarily a proper English word.

Syntax

```
newDocuments = normalizeWords(documents)
newWords = normalizeWords(words)
```

Description

`newDocuments = normalizeWords(documents)` stems each word in `documents` using the Porter stemmer.

`newWords = normalizeWords(words)` stems each word in `words`.

Examples

Stem Words in Document Array

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:

(1,1) 6 tokens: a strongli word collect of word
```

```
(2,1) 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 array
    "a"    "strongli"    "word"    "collect"    "of"    "word"
```

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`

Output Arguments

`newDocuments` — Output documents

`tokenizedDocument` array

Output documents, returned as a `tokenizedDocument` array.

`newWords` — Output words
string array | character vector | cell array of character vectors

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

See Also

`bagOfWords` | `lower` | `regexprep` | `replace` | `tokenizedDocument` | `upper`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

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:

(1,1) 70 tokens: fairest creatures desire increase thereby beautys rose might never die
(2,1) 71 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys fie
(3,1) 65 tokens: look thy glass tell face thou viewest time face form another whose fre
(4,1) 71 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys leg
(5,1) 61 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants
```

```
documents2 = documents(6:10)
```

```
documents2 =
  5x1 tokenizedDocument:

(1,1) 68 tokens: let winters ragged hand deface thee thy summer ere thou distilld make
(2,1) 64 tokens: lo orient gracious light lifts up burning head eye doth homage newappe
(3,1) 70 tokens: music hear why hearst thou music sadly sweets sweets war joy delights
(4,1) 70 tokens: fear wet widows eye thou consumst thy self single life ah thou issuele
(5,1) 69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt t
```

Append the second 5 sonnets to the first 5 sonnets.

```
newDocuments = documents1 + documents2
```

```
newDocuments =
  5x1 tokenizedDocument:

(1,1) 138 tokens: fairest creatures desire increase thereby beautys rose might never di
(2,1) 135 tokens: forty winters shall besiege thy brow dig deep trenches thy beautys fi
(3,1) 135 tokens: look thy glass tell face thou viewest time face form another whose fr
(4,1) 141 tokens: unthrifty loveliness why dost thou spend upon thy self thy beautys le
(5,1) 130 tokens: hours gentle work frame lovely gaze every eye doth dwell play tyrants
```

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

`tokenizedDocument`

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 0.0476809 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 |       0.07 | 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]
                      FitInfo: [1x1 struct]
```

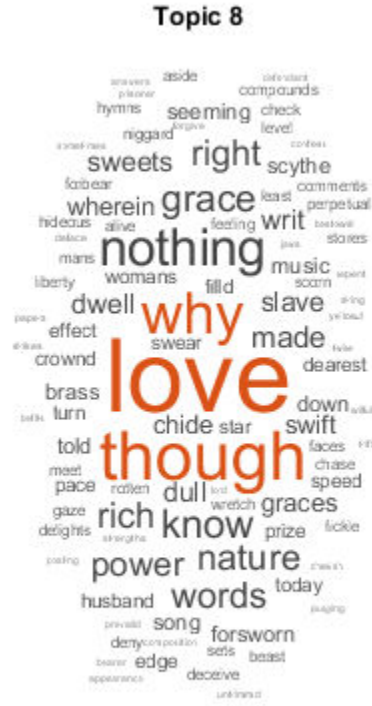
Predict the top topics for an array of new documents.

```
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 0.0904534 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 |       0.08 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.07 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.08 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.08 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.08 | 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]
                       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.

```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0473846 seconds.
=================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
```

```
=========================================================================================
|          0 |       0.04 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.09 | 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 |       0.07 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.07 | 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]
                       FitInfo: [1x1 struct]
```

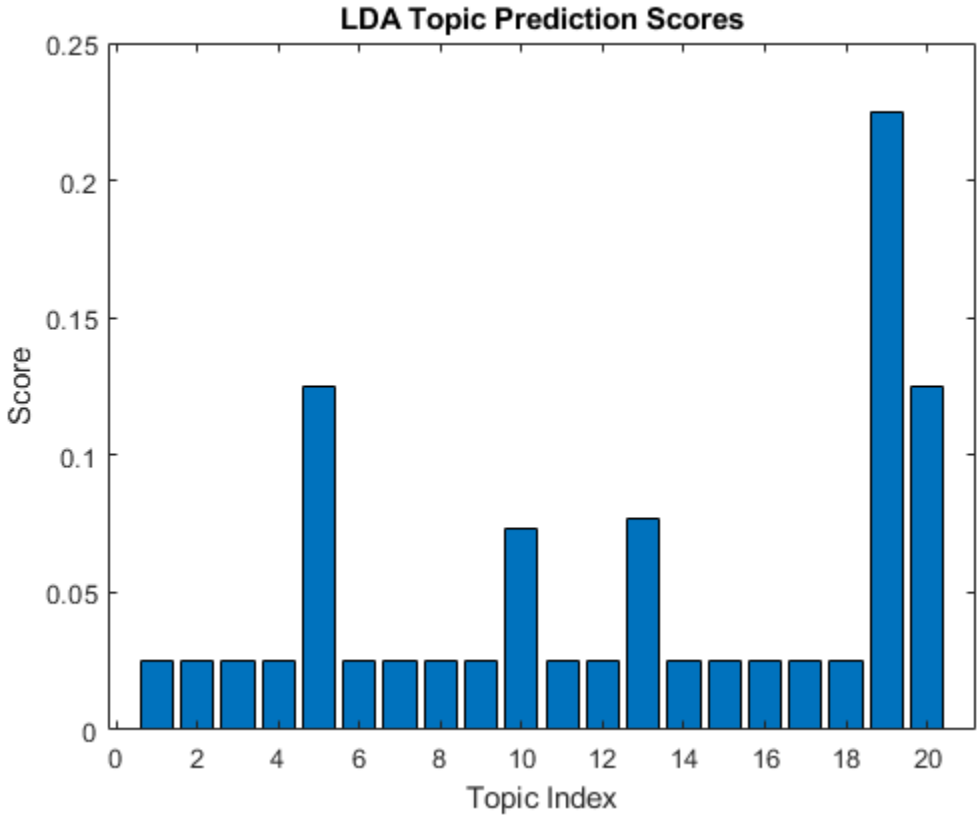
Predict the top topics for a new document. Specify the iteration limit to be 200.

```
newDocument = tokenizedDocument("what's in a name? a rose by any other name would smel
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    0.0
```

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 the n-grams as individual words.

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 single 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

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 filename using the wildcard "*" to find all filenames 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: {
                             ' ...\ib0BF173\8\tpd67a570d\textanalytics-ex39762425\weat
                             ' ...\ib0BF173\8\tpd67a570d\textanalytics-ex39762425\weat
                             ' ...\ib0BF173\8\tpd67a570d\textanalytics-ex39762425\weat
                              ... and 1 more
                             }
                UniformRead: 0
                   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 = 4x1 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`

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.

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

`fastTextWordEmbedding` | `ismember` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbedding` | `writeWordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

regexprep

Replace text in words of documents using regular expression

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”.

Syntax

```
newDocuments = regexprep(documents,expression,replace)
```

Description

`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:
```

```
(1,1) 6 tokens: an example of a short sentence
(2,1) 4 tokens: a second short sentence
```

```
expression = "^s(\w+)e$";
replace = "thing";
newDocuments = regexprep(documents,expression,replace)
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 6 tokens: an example of a short thing
(2,1) 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:

(1,1) 6 tokens: _n _x_mpl_ _f _ sh_rt s_nt_nc_
(2,1) 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:

(1,1) 2 tokens: I walk
```

```
(2,1) 2 tokens: they walked
(3,1) 3 tokens: we are walking


expression = "walk(\w*)";
replace = "ascend$1";
newDocuments = regexprep(documents,expression,replace)

newDocuments =
  3x1 tokenizedDocument:

(1,1) 2 tokens: I ascend
(2,1) 2 tokens: they ascended
(3,1) 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'`.
`[c1c2c3]`	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'`.
`[^c1c2c3]`	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'`.
`[c1-c2]`	Any character in the range of `c1` through `c2`	`'[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.
`\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.
`\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`.

Metacharacter	Description	Example
`\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
`\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	Matches the expression when it 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.

Quantifier	Matches the expression when it occurs...	Example
`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>'`
`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`.
`(?(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`.

Ordinal Token Operator	Description	Example
`\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`	`N`th 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

`bagOfWords` | `docfun` | `erasePunctuation` | `lower` | `normalizeWords` | `replace` | `tokenizedDocument` | `upper`

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”

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:

(1,1) 6 tokens: an example of a short sentence
(2,1) 0 tokens:
(3,1) 4 tokens: a second short sentence
(4,1) 0 tokens:
```

Remove the empty documents.

```
newDocuments = removeEmptyDocuments(documents)
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 6 tokens: an example of a short sentence
(2,1) 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: [4x7 double]
      Vocabulary: [1x7 string]
        NumWords: 7
    NumDocuments: 4
```

Remove the empty documents from the bag-of-words model.

```
newBag = removeEmptyDocuments(bag)
```

```
newBag =
  bagOfWords with properties:

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

(1,1) 6 tokens: an example of a short sentence
(2,1) 0 tokens:
(3,1) 4 tokens: a second short sentence
(4,1) 0 tokens:
```

Create a vector of labels.

```
labels = ["T"; "F"; "F"; "T"]
```

```
labels = 4x1 string array
    "T"
    "F"
    "F"
    "T"
```

Remove the empty documents and get the indices of the removed documents.

```
[newDocuments, idx] = removeEmptyDocuments(documents)
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 6 tokens: an example of a short sentence
(2,1) 4 tokens: a second short sentence


idx = 2×1

     2
     4
```

Remove the corresponding labels from `labels`.

```
labels(idx) = []
```

```
labels = 2x1 string array
    "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”

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)
```

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`.

`newBag = removeInfrequentNgrams(bag,count,'NgramLengths',lengths)` only removes n-grams with lengths specified by `lengths`.

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` | `removeNgrams` | `tokenizedDocument`

Introduced in R2018a

removeInfrequentWords

Remove words with low counts from bag-of-words model

Syntax

```
newBag = removeInfrequentWords(bag,count)
```

Description

`newBag = removeInfrequentWords(bag,count)` removes the words that appear at most `count` times in total from the bag-of-words model `bag`.

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

`bagOfWords` | `removeEmptyDocuments` | `removeLongWords` | `removeShortWords` | `removeWords` | `stopWords` | `tokenizedDocument`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

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

`bagOfWords` | `removeInfrequentWords` | `removeShortWords` | `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,idx)
```

Description

`newBag = removeNgrams(bag,ngrams)` removes the specified n-grams from the bag-of-n-grams model `bag`.

`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 array
    "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 array
    "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` | `tokenizedDocument`

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

`bagOfWords` | `removeInfrequentWords` | `removeLongWords` | `removeWords` | `stopWords` | `tokenizedDocument`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

removeWords

Remove selected words from documents or bag-of-words model

Syntax

```
newDocuments = removeWords(documents,words)
newDocuments = removeWords(documents,idx)

newBag = removeWords(bag,words)
newBag = removeWords(bag,idx)
```

Description

`newDocuments = removeWords(documents,words)` removes the specified words from `documents`.

`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,words)` removes the specified words from the bag-of-words model `bag`.

`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 Stop Words from Documents

Remove the stop words from an array of documents 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"]);
newDocuments = removeWords(documents,stopWords)
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 3 tokens: example short sentence
(2,1) 3 tokens: second short sentence
```

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:

(1,1) 3 tokens: I love MATLAB
(2,1) 3 tokens: I love MathWorks
```

View the vocabulary of `documents`.

```
documents.Vocabulary
```

```
ans = 1x4 string array
    "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:

(1,1) 1 tokens: love
(2,1) 2 tokens: love MathWorks
```

Alternatively, you can specify logical indices.

```
idx = logical([1 0 1 0]);
newDocuments = removeWords(documents,idx)
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 1 tokens: love
(2,1) 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 array
    "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

`bagOfWords` | `removeEmptyDocuments` | `removeInfrequentWords` | `removeLongWords` | `removeShortWords` | `stopWords` | `tokenizedDocument`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

replace

Find and replace substrings in documents

Syntax

```
newDocuments = replace(documents,old,new)
```

Description

`newDocuments = replace(documents,old,new)` replaces all occurrences of `old` in `documents` with `new`.

Examples

Replace Substrings in Documents

Replace words in a document array.

```
documents = tokenizedDocument([
    "an extreme example"
    "another extreme example"])
```

```
documents =
  2x1 tokenizedDocument:

(1,1) 3 tokens: an extreme example
(2,1) 3 tokens: another extreme example
```

```
newDocuments = replace(documents,"example","sentence")
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 3 tokens: an extreme sentence
```

```
(2,1) 3 tokens: another extreme sentence
```

Replace substrings of the words.

```
newDocuments = replace(documents,"ex","X-")
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 3 tokens: an X-treme X-ample
(2,1) 3 tokens: another X-treme X-ample
```

Input Arguments

`documents` — Input documents

`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

`old` — Substring to replace

string array | character vector | cell array of character vectors

Substring to replace, specified as a string array, character vector, or cell array of character vectors.

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

`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` | `docfun` | `lower` | `normalizeWords` | `replace` | `tokenizedDocument` | `upper`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

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 |       0.02 |            |  3.292e+03 |         1.000 |             0 |
|          1 |       0.02 | 1.4970e-01 |  1.147e+03 |         1.000 |             0 |
|          2 |       0.01 | 7.1229e-03 |  1.091e+03 |         1.000 |             0 |
|          3 |       0.02 | 8.1261e-03 |  1.031e+03 |         1.000 |             0 |
|          4 |       0.02 | 8.8626e-03 |  9.703e+02 |         1.000 |             0 |
|          5 |       0.03 | 8.5486e-03 |  9.154e+02 |         1.000 |             0 |
|          6 |       0.01 | 7.4632e-03 |  8.703e+02 |         1.000 |             0 |
|          7 |       0.02 | 6.0480e-03 |  8.356e+02 |         1.000 |             0 |
|          8 |       0.02 | 4.5955e-03 |  8.102e+02 |         1.000 |             0 |
|          9 |       0.01 | 3.4068e-03 |  7.920e+02 |         1.000 |             0 |
|         10 |       0.01 | 2.5353e-03 |  7.788e+02 |         1.000 |             0 |
|         11 |       0.02 | 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 |       0.01 | 9.1253e-04 |  7.524e+02 |         1.079 |             7 |
|         15 |       0.02 | 7.5878e-04 |  7.486e+02 |         1.039 |             6 |
```

```
|         16 |       0.03 | 6.6181e-04 |  7.454e+02 |         1.004 |             6 |
|         17 |       0.02 | 6.0400e-04 |  7.424e+02 |         0.974 |             6 |
|         18 |       0.02 | 5.6244e-04 |  7.396e+02 |         0.948 |             6 |
|         19 |       0.02 | 5.0548e-04 |  7.372e+02 |         0.926 |             5 |
|         20 |       0.01 | 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 |       0.02 | 3.4941e-04 |  7.334e+02 |         0.887 |             5 |
|         22 |       0.03 | 2.9495e-04 |  7.320e+02 |         0.871 |             5 |
|         23 |       0.01 | 2.6300e-04 |  7.307e+02 |         0.857 |             5 |
|         24 |       0.01 | 2.5200e-04 |  7.295e+02 |         0.844 |             4 |
|         25 |       0.01 | 2.4150e-04 |  7.283e+02 |         0.833 |             4 |
|         26 |       0.01 | 2.0549e-04 |  7.273e+02 |         0.823 |             4 |
|         27 |       0.02 | 1.6441e-04 |  7.266e+02 |         0.813 |             4 |
|         28 |       0.01 | 1.3256e-04 |  7.259e+02 |         0.805 |             4 |
|         29 |       0.01 | 1.1094e-04 |  7.254e+02 |         0.798 |             4 |
|         30 |       0.01 | 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]
                     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 |       0.02 | 8.0569e-05 |  7.246e+02 |         0.785 |             3 |
|         32 |       0.02 | 7.4692e-05 |  7.242e+02 |         0.779 |             3 |
|         33 |       0.01 | 6.9802e-05 |  7.239e+02 |         0.774 |             3 |
|         34 |       0.01 | 6.1154e-05 |  7.236e+02 |         0.770 |             3 |
|         35 |       0.01 | 5.3163e-05 |  7.233e+02 |         0.766 |             3 |
|         36 |       0.01 | 4.7807e-05 |  7.231e+02 |         0.762 |             3 |
|         37 |       0.01 | 4.1820e-05 |  7.229e+02 |         0.759 |             3 |
|         38 |       0.03 | 3.6237e-05 |  7.227e+02 |         0.756 |             3 |
|         39 |       0.02 | 3.1819e-05 |  7.226e+02 |         0.754 |             2 |
|         40 |       0.02 | 2.7772e-05 |  7.224e+02 |         0.751 |             2 |
|         41 |       0.01 | 2.5238e-05 |  7.223e+02 |         0.749 |             2 |
|         42 |       0.01 | 2.2052e-05 |  7.222e+02 |         0.747 |             2 |
|         43 |       0.02 | 1.8471e-05 |  7.221e+02 |         0.745 |             2 |
|         44 |       0.02 | 1.5638e-05 |  7.221e+02 |         0.744 |             2 |
|         45 |       0.02 | 1.3735e-05 |  7.220e+02 |         0.742 |             2 |
|         46 |       0.02 | 1.2298e-05 |  7.219e+02 |         0.741 |             2 |
|         47 |       0.01 | 1.0905e-05 |  7.219e+02 |         0.739 |             2 |
|         48 |       0.01 | 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]
       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 the n-grams as individual words.

`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 single 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 $\alpha = \alpha_0 \begin{pmatrix} p_1 & p_2 & \cdots & p_K \end{pmatrix}$ 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.

`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”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

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 array
    "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.

See Also

`erasePunctuation` | `eraseTags` | `eraseURLs` | `split` | `splitlines` | `tokenizedDocument`

Introduced in R2018a

stopWords

List of stop words

Syntax

```
words = stopWords
```

Description

`words = stopWords` returns a string array of common words which can be removed from documents before analysis. For an example showing how to remove stop words from documents, see “Remove Stop Words from Documents” on page 1-236.

Examples

Remove Stop Words from Documents

Remove the stop words from an array of documents 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"]);
newDocuments = removeWords(documents,stopWords)
```

```
newDocuments =
  2x1 tokenizedDocument:

(1,1) 3 tokens: example short sentence
(2,1) 3 tokens: second short sentence
```

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
```

See Also

`bagOfWords` | `removeInfrequentWords` | `removeLongWords` | `removeShortWords` | `removeWords` | `tokenizedDocument` | `topLevelDomains`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

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 array
    "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

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.

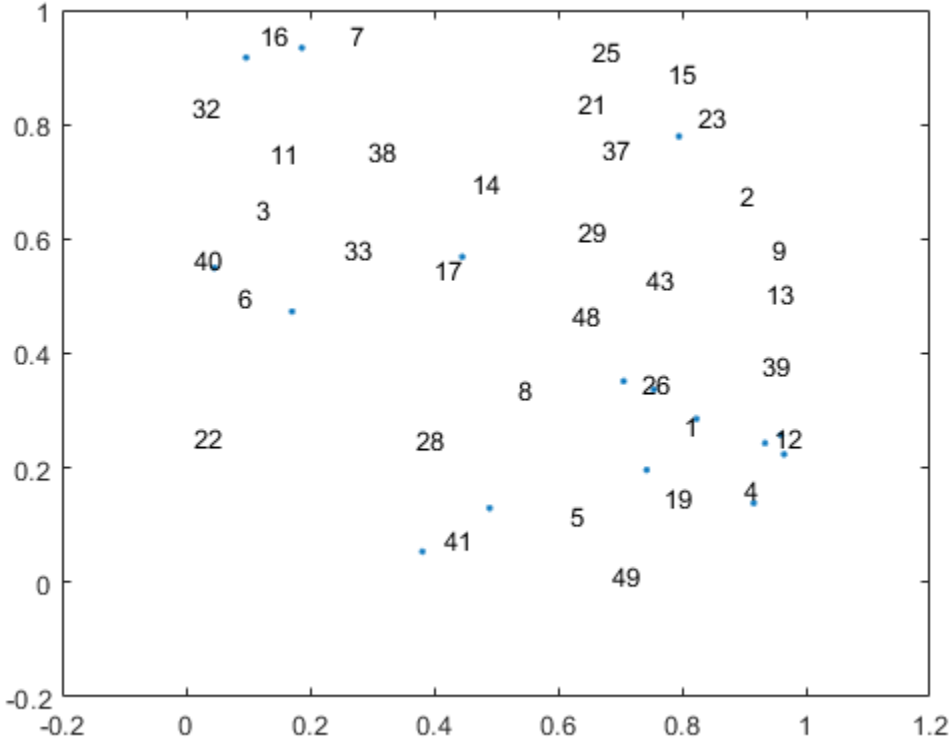
Examples

Create Text Scatter Plot

Plot a string array of numbers at random points on a text scatter plot.

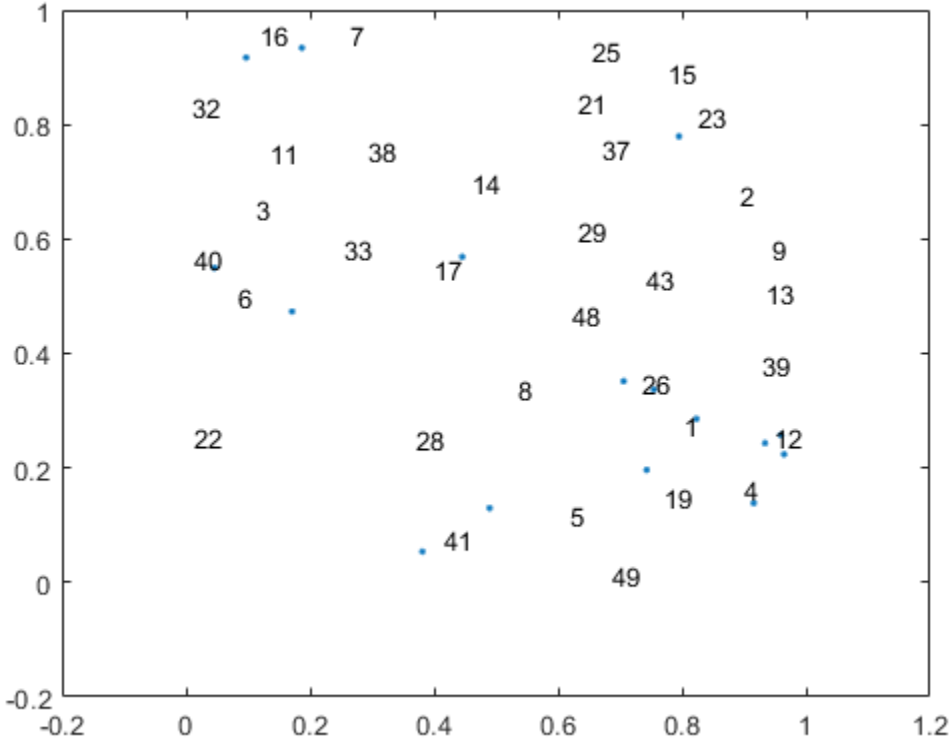
```
x = rand(50,1);
y = rand(50,1);
str = string(1:50);
```

```
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.

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]
```

Convert the first 500 words to vectors using `word2vec`. `V` is a matrix of word vectors of length 50.

```
words = emb.Vocabulary(1:500);
V = word2vec(emb,words);
size(V)
```

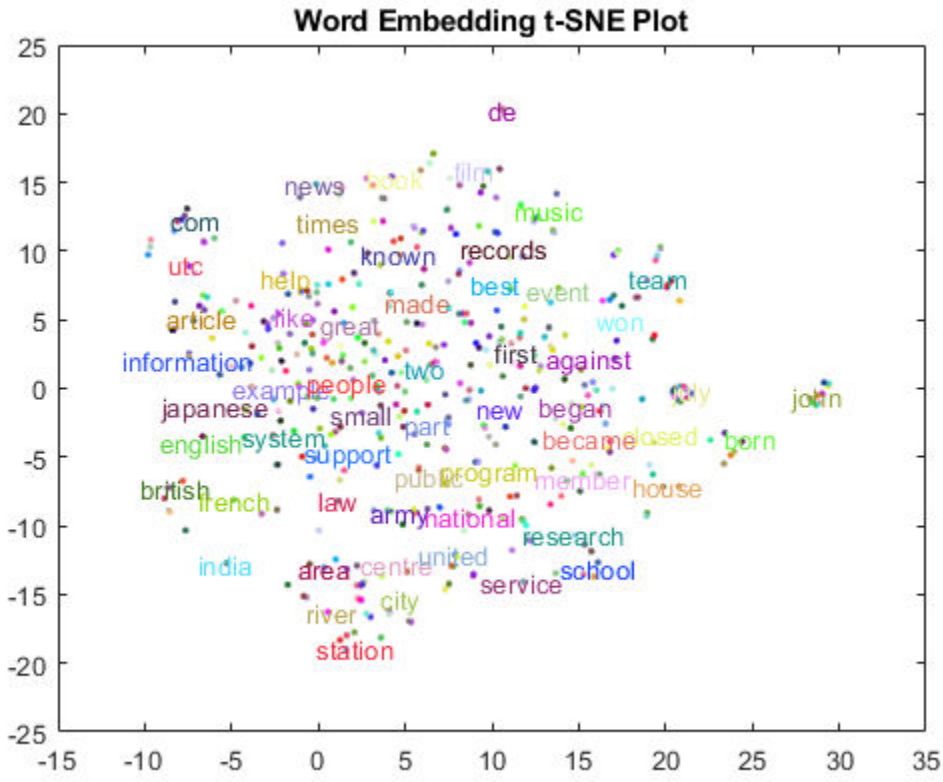
```
ans = 1×2

   500    50
```

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 single 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

`textscatter3` | `wordcloud`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Visualize Text Data Using Word Clouds”
“Prepare Text Data for Analysis”
“Analyze Text Data Using Topic Models”

Introduced in R2017b

textscatter3

3-D scatter plot of text

Syntax

```
ts = textscatter3(x,y,zstr)
ts = textscatter3(xyz,str)
ts = textscatter3(ax, ___ )
ts = textscatter3( ___ ,Name,Value)
```

Description

`ts = textscatter3(x,y,zstr)` 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.

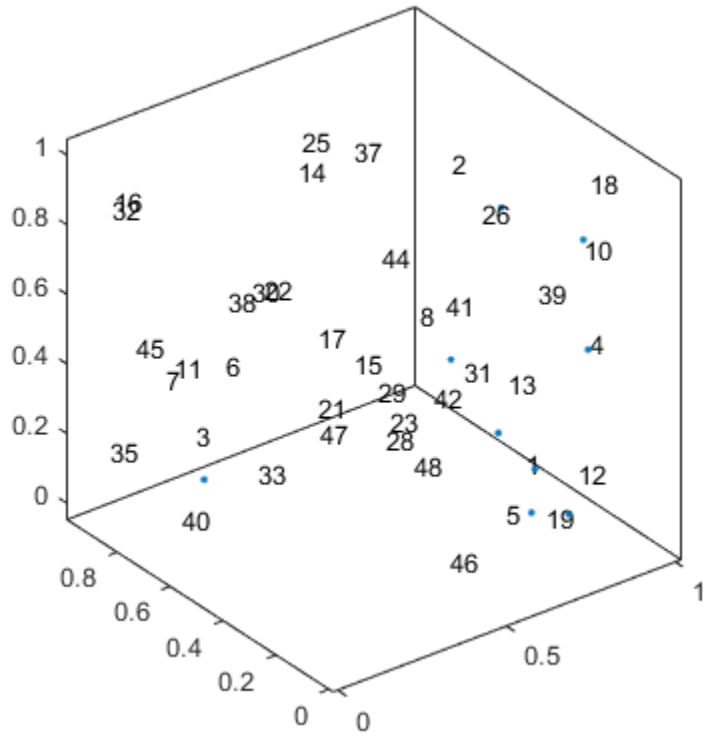
Examples

Create 3-D Text Scatter Plot

Plot a string array of numbers at random points on a 3-D text scatter plot.

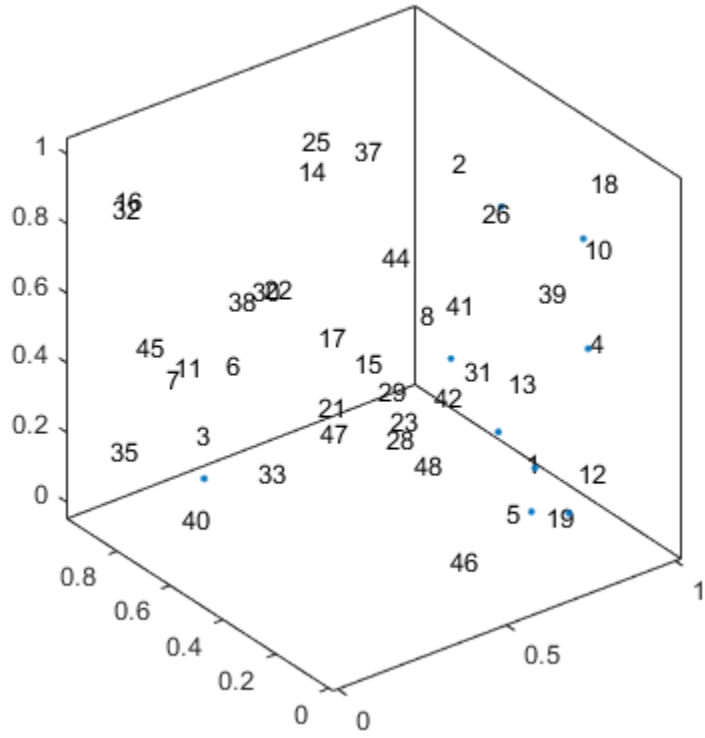
```
x = rand(50,1);
y = rand(50,1);
z = rand(50,1);
```

```
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.

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]
```

Convert the first 500 words to vectors using `word2vec`. `V` is a matrix of word vectors of length 50.

```
words = emb.Vocabulary(1:500);
V = word2vec(emb,words);
size(V)
```

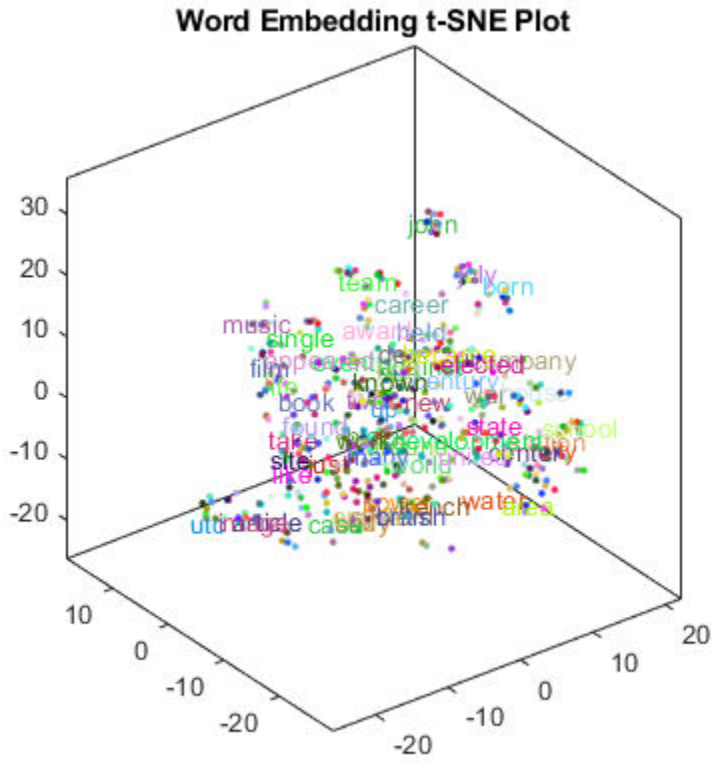
```
ans = 1×2

   500    50
```

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]`

`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 single 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

`textscatter` | `wordcloud`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Visualize Text Data Using Word Clouds”
“Prepare Text Data for Analysis”
“Analyze Text Data Using Topic Models”

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.

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]. Alternatively, you can specify some common colors by name. This table lists the long and short color name options and the equivalent RGB triplet values.

Option	Description	Equivalent RGB Triplet
'red' or 'r'	Red	[1 0 0]
'green' or 'g'	Green	[0 1 0]
'blue' or 'b'	Blue	[0 0 1]
'yellow' or 'y'	Yellow	[1 1 0]
'magenta' or 'm'	Magenta	[1 0 1]
'cyan' or 'c'	Cyan	[0 1 1]
'white' or 'w'	White	[1 1 1]
'black' or 'k'	Black	[0 0 0]

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

The object has no children. You cannot set this property.

`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" (MATLAB).

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}`

`UIContextMenu` — Context menu
`uicontextmenu` object

Context menu, specified as a `uicontextmenu` 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'`.

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” (MATLAB).
  - 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" (MATLAB).

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 objectMATLAB 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" (MATLAB).

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.

See Also

`textscatter` | `textscatter3` | `wordcloud`

Topics

"Visualize Word Embeddings Using Text Scatter Plots"
"Visualize Text Data Using Word Clouds"
"Prepare Text Data for Analysis"
"Analyze Text Data Using Topic Models"

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    2.4
         0         0         0         0         0    4.5287         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0
         0         0         0         0         0    2.2644         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    2.2644         0         0
         0         0    2.7344         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    2.4
         0         0         0         0         0    4.5287         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0
         0         0         0         0         0    2.2644         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    2.2644         0         0
         0         0    2.7344         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    2.4
         0         0         0         0         0    2.2644         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0    2.2644         0         0
         0         0         0         0         0    2.2644         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    2.2644         0         0
         0         0    2.7344         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 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.

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 single 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 set inverse document frequency factor
`'normal'` (default) | `'unary'` | `'smooth'` | `'max'` | `'probabilistic'`

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

- `'normal'` – Set the IDF factor to `log(N/NT)`.
- `'unary'` – Set the IDF factor to `1`.
- `'smooth'` – Set the IDF factor to `log(1+N/NT)`.
- `'max'` – Set the IDF factor to `log(1+max(NT)/NT)`.
- `'probabilistic'` – Set the IDF factor to `log((N-NT)/NT)`.

where `N` is the number of documents in the `bag`, and `NT` is the number of documents containing each term which is equivalent to `sum(bag.Counts)`.

Example: `'IDFWeight','smooth'`

Data Types: `char`

`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`.

See Also

`bagOfNgrams` | `bagOfWords` | `encode` | `topkngrams` | `topkwords`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

tokenizedDocument

Array of tokenized documents

Description

A tokenized document is a document represented as a collection of words (also known as tokens) which is used for analysis. `tokenizedDocument` arrays allows you to perform word-level processing tasks such as removing words using `removeWords` and stemming words using `normalizeWords`.

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 `str` and returns an array of tokenized documents.

`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 short document'`

Example: `{'an example of a short document';'a second short document'}`

Example: `{"an example of a short document";"a second short document"}`

If the input text has already been split into words, then `str` must be a cell array and you must specify `'TokenizeMethod'` to be `'none'`.

If `str` contains a single document, then it must be a row of character vectors, or contain a single string array of words. Otherwise, `str` must contain row vectors of strings.

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`

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single 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'` (default) | `'none'`

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

- `'unicode'` – Tokenize input text into words. If `str` is a cell array, then the elements of `str` must be string scalars, or character vectors.

- `'none'` – Do not tokenize the input text. `str` must be a cell array.

If `'TokenizeMethod'` is `'none'`, then the function `tokenDetails` returns an empty table. To add tokens with document and sentence numbers to the table, use `addSentenceDetails`.

If `str` contains a single document, then it must be a row of character vectors, or contain a single string array of words. Otherwise, `str` must contain row vectors of strings.

Example: `'none'`

`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 `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.

If `DetectPatterns` is `'none'`, then the function does not detect any complex tokens 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`

`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 the `'DetectPatterns'` is `'all'` or contains `'web-address'`.

The function, by default, uses the output of `topLevelDomains`.

Example: `'TopLevelDomains',["com" "net" "org"]`

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

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

Manipulation

erasePunctuation	Erase punctuation from text and documents
removeWords	Remove selected words from documents or bag-of-words model
normalizeWords	Reduce words to common stems using the Porter stemmer
removeEmptyDocuments	Remove empty documents from tokenized document array, bag-of-words model, or bag-of-n-grams model
addSentenceDetails	Add sentence numbers to documents
lower	Convert documents to lowercase
upper	Convert documents to uppercase
plus	Append documents
replace	Find and replace substrings in documents
docfun	Apply function to words in documents
regexprep	Replace text in words of documents using regular expression

Exploration

tokenDetails	Details of tokens in tokenized document array
doclength	Length of documents in document array
context	Search documents for word occurrences in context

Export

writeTextDocument Write documents to text file

Conversion

joinwords Convert documents to string by joining words
doc2cell Convert documents to cell array of string vectors
string Convert scalar document to string vector

Examples

Create Tokenized Documents

Create tokenized documents from a string array.

```
str = [
    "an example of a short sentence"
    "a second short sentence"]
```

```
str = 2x1 string array
    "an example of a short sentence"
    "a second short sentence"
```

```
documents = tokenizedDocument(str)
```

```
documents =
  2x1 tokenizedDocument:

(1,1) 6 tokens: an example of a short sentence
(2,1) 4 tokens: a second short sentence
```

Detect Complex Tokens

Create a tokenized document of the string `str`. By default, the function treats the hashtag `"#MATLAB"` and the web address `"https://www.mathworks.com/help"` as single tokens.

```
str = "Learn how to analyze text in #MATLAB, see https://www.mathworks.com/help/";
document = tokenizedDocument(str)

document =
  tokenizedDocument:

   10 tokens: Learn how to analyze text in #MATLAB , see https://www.mathworks.com/help
```

To detect hashtags only as complex tokens, specify the `'DetectPatterns'` option to be `'hashtag'` only. The function then tokenizes the web address `"https://www.mathworks.com/help"` into multiple tokens.

```
document = tokenizedDocument(str,'DetectPatterns','hashtag')

document =
  tokenizedDocument:

   21 tokens: Learn how to analyze text in #MATLAB , see https : / / www . mathworks .
```

Remove Stop Words from Documents

Remove the stop words from an array of documents 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"]);
newDocuments = removeWords(documents,stopWords)

newDocuments =
  2x1 tokenizedDocument:

(1,1) 3 tokens: example short sentence
(2,1) 3 tokens: second short sentence
```

Stem Words in Document Array

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:

(1,1) 6 tokens: a strongli word collect of word
(2,1) 4 tokens: anoth collect of word
```

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 array
    "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"
```

- “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”

See Also

`addSentenceDetails` | `bagOfWords` | `context` | `doc2cell` | `docfun` | `doclength` | `erasePunctuation` | `joinwords` | `lower` | `normalizeWords` | `plus` | `regexprep` | `removeEmptyDocuments` | `removeWords` | `replace` | `string` | `tokenDetails` | `topLevelDomains` | `upper` | `writeTextDocument`

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”

Introduced in R2017b

tokenDetails

Details of tokens in tokenized document array

Syntax

```
details = tokenDetails(documents)
```

Description

`details = tokenDetails(documents)` returns a table of token details for the tokens in `documents`.

Examples

View Token Details of Documents

Create a tokenized document from the text in `exampleSonnet1.txt`.

```
filename = "exampleSonnet1.txt";
str = extractFileText(filename);
document = tokenizedDocument(str);
```

View the token details of the first few tokens.

```
details = tokenDetails(document);
head(details)
```

```
ans=8×4 table
       Token       DocumentNumber    LineNumber       Type
    ___________    ______________    __________    ___________

    "From"              1               1         letters
    "fairest"           1               1         letters
    "creatures"         1               1         letters
    "we"                1               1         letters
```

```
    "desire"            1              1         letters
    "increase"          1              1         letters
    ","                 1              1         punctuation
    "That"              1              2         letters
```

View the token details of the second line of the document.

```
details(details.LineNumber == 2,:)
```

```
ans=8×4 table
      Token       DocumentNumber    LineNumber       Type
    __________    ______________    __________    __________

    "That"              1               2         letters
    "thereby"           1               2         letters
    "beauty's"          1               2         other
    "rose"              1               2         letters
    "might"             1               2         letters
    "never"             1               2         letters
    "die"               1               2         letters
    ","                 1               2         punctuation
```

Add Sentence Details to Documents

Create a tokenized document from the text in `exampleSonnet1.txt`.

```
filename = "exampleSonnet1.txt";
str = extractFileText(filename);
document = tokenizedDocument(str)
```

```
document =
  tokenizedDocument:

   124 tokens: From fairest creatures we desire increase , That thereby beauty's rose m
```

View the token details of the first 15 tokens.

```
details = tokenDetails(document);
head(details,15)
```

```
ans=15×4 table
       Token       DocumentNumber    LineNumber       Type
```

```
    ___________    ______________    __________    ___________

    "From"              1               1          letters
    "fairest"           1               1          letters
    "creatures"         1               1          letters
    "we"                1               1          letters
    "desire"            1               1          letters
    "increase"          1               1          letters
    ","                 1               1          punctuation
    "That"              1               2          letters
    "thereby"           1               2          letters
    "beauty's"          1               2          other
    "rose"              1               2          letters
    "might"             1               2          letters
    "never"             1               2          letters
    "die"               1               2          letters
    ","                 1               2          punctuation
```

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 15 tokens.

```
document = addSentenceDetails(document);
details = tokenDetails(document);
head(details,15)
```

```
ans=15×5 table
       Token       DocumentNumber    SentenceNumber    LineNumber       Type
    ___________    ______________    ______________    __________    ___________

    "From"              1                 1                1          letters
    "fairest"           1                 1                1          letters
    "creatures"         1                 1                1          letters
    "we"                1                 1                1          letters
    "desire"            1                 1                1          letters
    "increase"          1                 1                1          letters
    ","                 1                 1                1          punctuation
    "That"              1                 1                2          letters
    "thereby"           1                 1                2          letters
    "beauty's"          1                 1                2          other
    "rose"              1                 1                2          letters
    "might"             1                 1                2          letters
    "never"             1                 1                2          letters
    "die"               1                 1                2          letters
```

```
    ","                1                1              2          punctuation
```

Input Arguments

`documents` — Input documents

`tokenizedDocument` array

Input documents, specified as a `tokenizedDocument` array.

Output Arguments

`details` — Table of token details

table

Table of token details. `T` has the following variables:

Name	Description
`Token`	Token text, specified as a string scalar.
`DocumentNumber`	Index of document that the token belongs to.
`LineNumber`	Line number of token in document.

Name	Description
`Type`	The type of token, specified as one of the following: • `'letters'` – consists of letter characters only • `'digits'` – consists of digits only • `'punctuation'` – consists 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) • `'other'` – does not belong to above types
`SentenceNumber`	Sentence number of token in document. To get this variable, you must first use `addSentenceDetails`.

If the input documents were tokenized with the `'TokenizeMethod'` option set to `'none'`, then the function returns an empty table. To add tokens with document and sentence numbers to the table, use `addSentenceDetails`.

See Also

`addSentenceDetails` | `context` | `erasePunctuation` | `removeWords` | `tokenizedDocument`

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`.

`tbl = topkwords(bag,k)` returns a table of the `k` words with the largest word counts.

`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.

```
numTopics = 20;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.0747008 seconds.
=================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=================================================================================
|          0 |       0.09 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
```

```
|          2 |       0.07 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.08 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.07 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.08 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.08 | 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]
                       FitInfo: [1x1 struct]
```

Find the top 20 words of the first topic.

```
k = 20;
topicIdx = 1;
T = topkwords(mdl,k,topicIdx)
```

```
T=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.

```
T = topkwords(mdl,k,topicIdx,'Scaling','inversemean')
```

```
T=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(T.Word,T.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 single 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.

`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`.

Only has an effect if the input data is a bag-of-words model.

Data Types: `logical`

`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`.

Only has an effect if the input data is an LDA model.

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

`bagOfWords` | `encode` | `ldaModel` | `tfidf` | `topkngrams`

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”
“Analyze Text Data Using Topic Models”

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`.

`tbl = topkngrams(bag,k)` lists the `k` most frequently seen n-grams in the bag-of-n-grams model `bag`.

`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 single 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]`

`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` | `encode` | `tfidf` | `topkwords`

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 usually used to detect web addresses in strings. Reshape the output for readability.

```
domains = topLevelDomains;
reshape(domains, [], 5)
```

```
ans = 51x5 string array
    "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"
"bh"    "fr"    "lc"    "pl"    "ua"
"bi"    "ga"    "li"    "pm"    "ug"
"bj"    "gd"    "lk"    "pn"    "uk"
"bl"    "ge"    "lr"    "pr"    "um"
"bm"    "gf"    "ls"    "ps"    "us"
"bn"    "gg"    "lt"    "pt"    "uy"
"bo"    "gh"    "lu"    "pw"    "uz"
"br"    "gi"    "lv"    "py"    "va"
"bs"    "gl"    "ly"    "qa"    "vc"
"bt"    "gm"    "ma"    "re"    "ve"
"bv"    "gn"    "mc"    "ro"    "vg"
"bw"    "gp"    "md"    "rs"    "vi"
"by"    "gq"    "me"    "ru"    "vn"
"bz"    "gr"    "mf"    "rw"    "vu"
"ca"    "gs"    "mg"    "sa"    "wf"
"cc"    "gt"    "mh"    "sb"    "ws"
"cd"    "gu"    "mk"    "sc"    "ye"
"cf"    "gw"    "ml"    "sd"    "yt"
"cg"    "gy"    "mm"    "se"    "za"
"ch"    "hk"    "mn"    "sg"    "zm"
"ci"    "hm"    "mo"    "sh"    "zw"
```

See Also

`bagOfWords` | `eraseURLs` | `stopWords` | `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

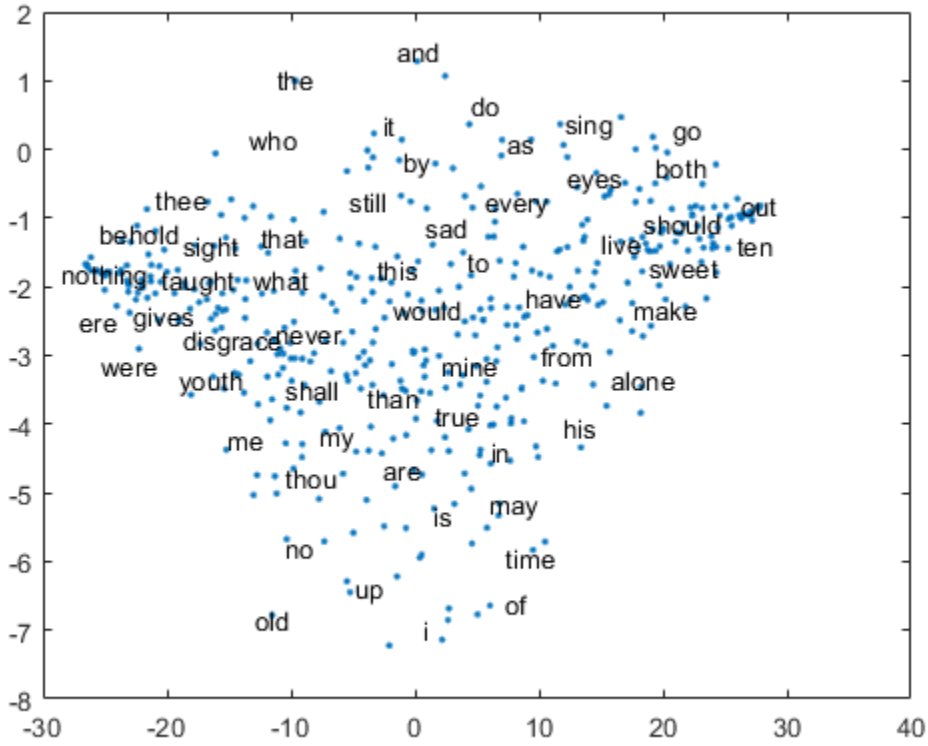
Train a word embedding of dimension 20 using the example text file `exampleSonnetsDocuments.txt`. This file contains preprocessed versions of Shakespeare's sonnets, with one sonnet per line and words separated by a space.

```
filename = "exampleSonnetsDocuments.txt";
emb = trainWordEmbedding(filename)
```

```
Training: 100% Loss: 2.66767  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);
```

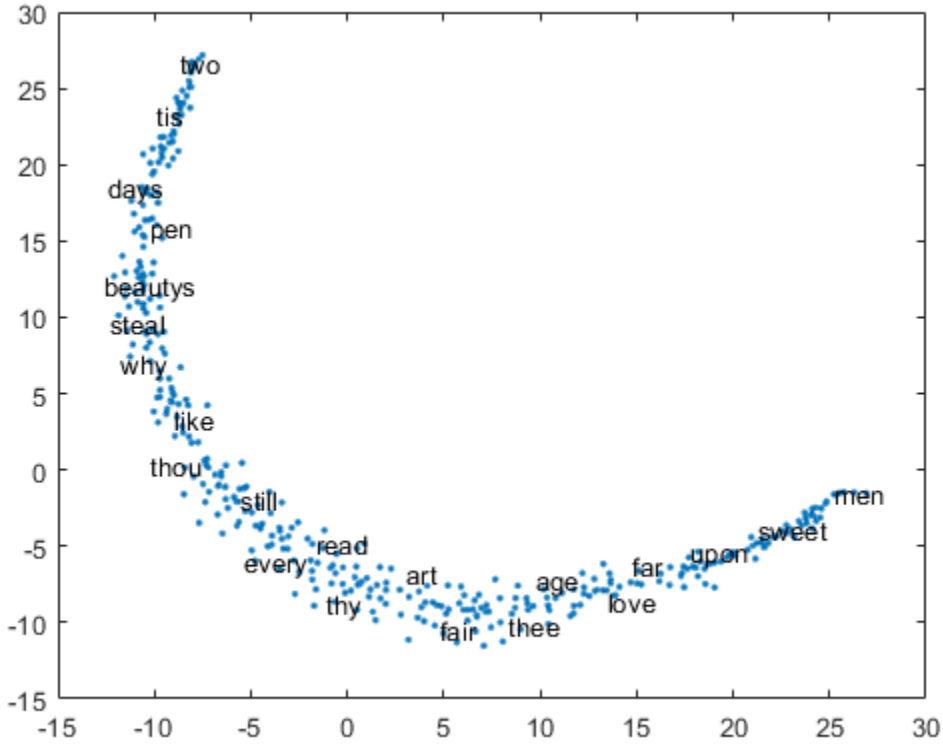
Train a word embedding using `trainWordEmbedding`.

```
emb = trainWordEmbedding(documents)
```

```
Training: 100% Loss: 2.74016  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);
```

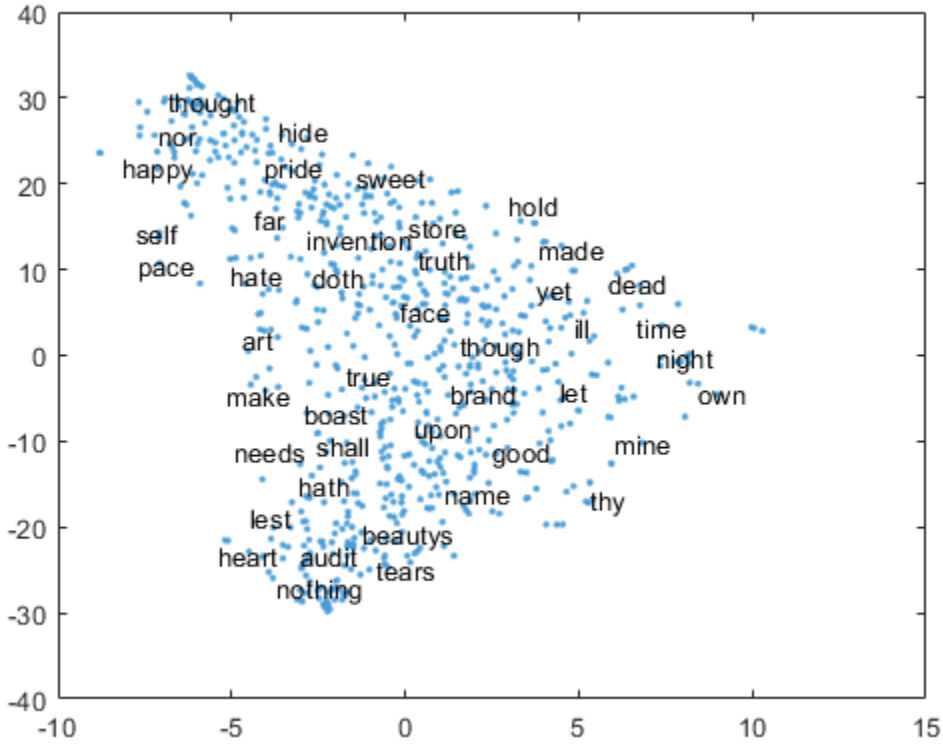
Specify the word embedding dimension to be 50. To reduce the number of words discarded by the model, set `'MinCount'` to 3. To train for longer, set the number of epochs to 10.

```
emb = trainWordEmbedding(documents, ...
    'Dimension',50, ...
    'MinCount',3, ...
    'NumEpochs',10)
```

```
Training: 100% Loss: 0        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 single 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) | nonnegative 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.

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

`fastTextWordEmbedding` | `ismember` | `readWordEmbedding` | `tokenizedDocument` | `vec2word` | `word2vec` | `wordEmbedding` | `writeWordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

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    0.2
    7.3069   -2.3578    1.8359   -2.3442   -1.5776   -2.0310   -0.7948    1.3411   -1.1
    7.1056   -2.3508   -2.8837   -1.0688   -0.3462   -0.6962   -0.0334   -0.0472    0.4
    8.6292   -3.0471   -0.8512   -0.4356   -0.3055    0.4671    1.4219   -0.8454   -0.8
    1.0434    1.7490    0.8703   -2.2315   -1.1221    0.2848    2.0522   -0.6975    1.7
    6.8358   -2.0806   -3.3798   -1.0452   -0.2075    2.0970    0.4477    0.2080    0.9
    2.3847    0.3923   -0.4323   -1.5340    0.4023   -1.0396    1.0326    0.3776    0.2
    3.7925   -0.3941   -4.4610   -0.4930    0.4651    0.3404    0.5493    0.1470    0.5
    4.6522    0.7188   -1.1787   -0.8996    0.3360    0.4531    0.1935    0.3328   -0.8
    8.8218   -0.8168   -2.5101    1.1197   -0.8673   -1.2336   -0.0768    0.1943   -0.7
```

Transform Documents into LDA Topic Mixtures

Load the `sonnetsDocuments` data and create a bag-of-words model. `sonnetsDocuments` returns a `tokenizedDocument` array of preprocessed versions of Shakespeare's sonnets.

To use the example file `sonnetsDocuments.m`, add the example folder to the path. To reproduce the results, set `rng` to `'default'`.

```
exampleFolder = genpath(fullfile(matlabroot,'examples','textanalytics'));
addpath(exampleFolder)
rng('default')
```

Load the `sonnetsDocuments` data and create a bag-of-words model.

```
documents = sonnetsDocuments;
bag = bagOfWords(documents)
```

```
bag =
  bagOfWords with properties:

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

Fit an LDA model with five topics.

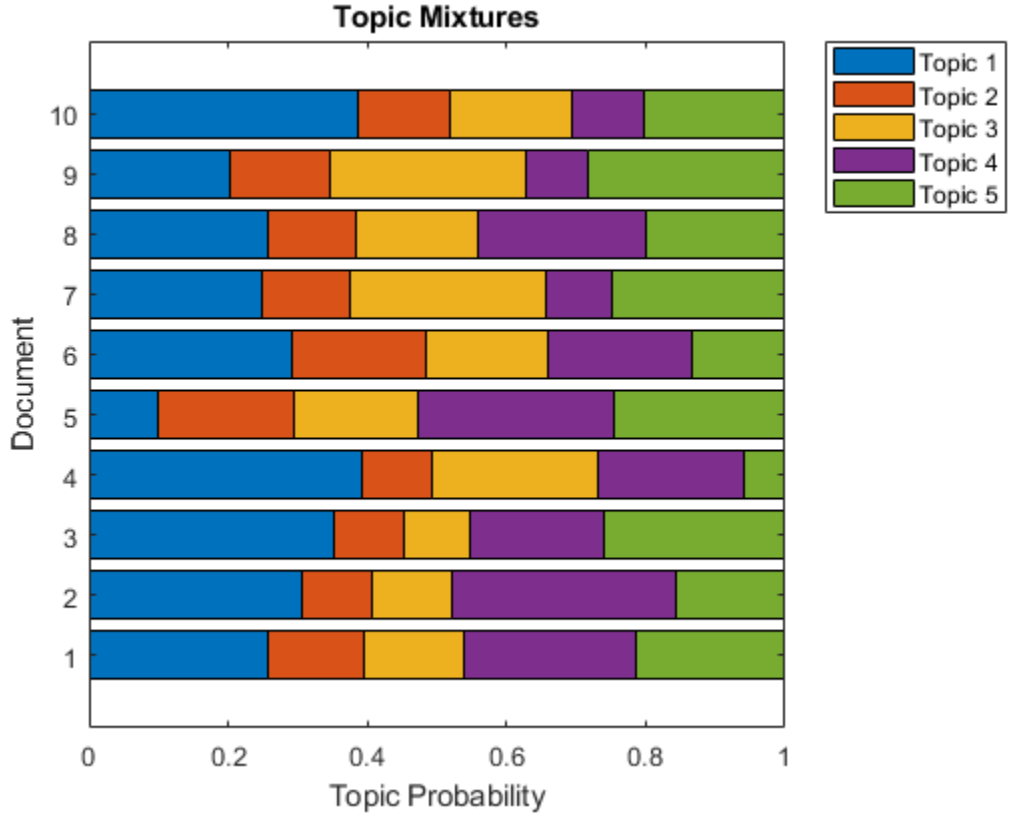
```
numTopics = 5;
mdl = fitlda(bag,numTopics)
```

```
Initial topic assignments sampled in 0.065897 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.02 |            |  1.212e+03 |         1.250 |             0 |
|          1 |       0.04 | 1.2300e-02 |  1.112e+03 |         1.250 |             0 |
|          2 |       0.03 | 1.3254e-03 |  1.102e+03 |         1.250 |             0 |
|          3 |       0.03 | 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]
                       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', 'bestoutside')
```



Remove the example folder from the path using `rmpath`.

```
rmpath(exampleFolder)
```

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 0.0795471 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       0.05 |            |  1.159e+03 |         5.000 |             0 |
|          1 |       0.07 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.07 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.07 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.08 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.06 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.07 | 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]
```

```
    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    0.0
    0.0711    0.0544    0.0116    0.0044    0.0033    0.0033    0.0431    0.0053    0.0
    0.0293    0.0482    0.1078    0.0322    0.0036    0.0036    0.0464    0.0036    0.0
    0.0055    0.0962    0.2403    0.0033    0.0296    0.1613    0.0164    0.0955    0.0
    0.0341    0.0224    0.0341    0.0645    0.0948    0.0038    0.0189    0.1099    0.0
    0.0445    0.0035    0.1167    0.0034    0.0446    0.0583    0.1268    0.0169    0.0
    0.1720    0.0764    0.0090    0.0180    0.0325    0.1213    0.0036    0.0036    0.0
    0.0043    0.0033    0.1248    0.0033    0.0299    0.0033    0.0690    0.1699    0.0
    0.0412    0.0387    0.0555    0.0165    0.0166    0.0433    0.0033    0.0038    0.0
    0.0362    0.0035    0.1117    0.0304    0.0034    0.1248    0.0439    0.0340    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 the n-grams as individual words.

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 single 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:

(1,1) 6 tokens: An Example of a Short Sentence
(2,1) 4 tokens: A Second Short Sentence
```

```
newDocuments = upper(documents)
```

```
newDocuments =
  2x1 tokenizedDocument:
```

```
(1,1) 6 tokens: AN EXAMPLE OF A SHORT SENTENCE
(2,1) 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

`bagOfWords` | `docfun` | `lower` | `normalizeWords` | `regexprep` | `replace` | `tokenizedDocument`

Topics

“Prepare Text Data for Analysis”
“Create Simple Text Model for Classification”

Introduced in R2017b

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

Explore Word Embedding

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]
```

Map the words "king", "man", and "woman" to vectors using `word2vec`.

```
king = word2vec(emb,"king");
man = word2vec(emb,"man");
woman = word2vec(emb,"woman");
```

Map the vector `king - man + woman` to a word using `vec2word`.

```
word = vec2word(emb,king - man + woman)
```

```
word =
"queen"
```

Find Closest Words to Vector

Find the top five closest words to a word embedding vector and their distances.

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]
```

Map the words "king", "man", and "woman" to vectors using word2vec.

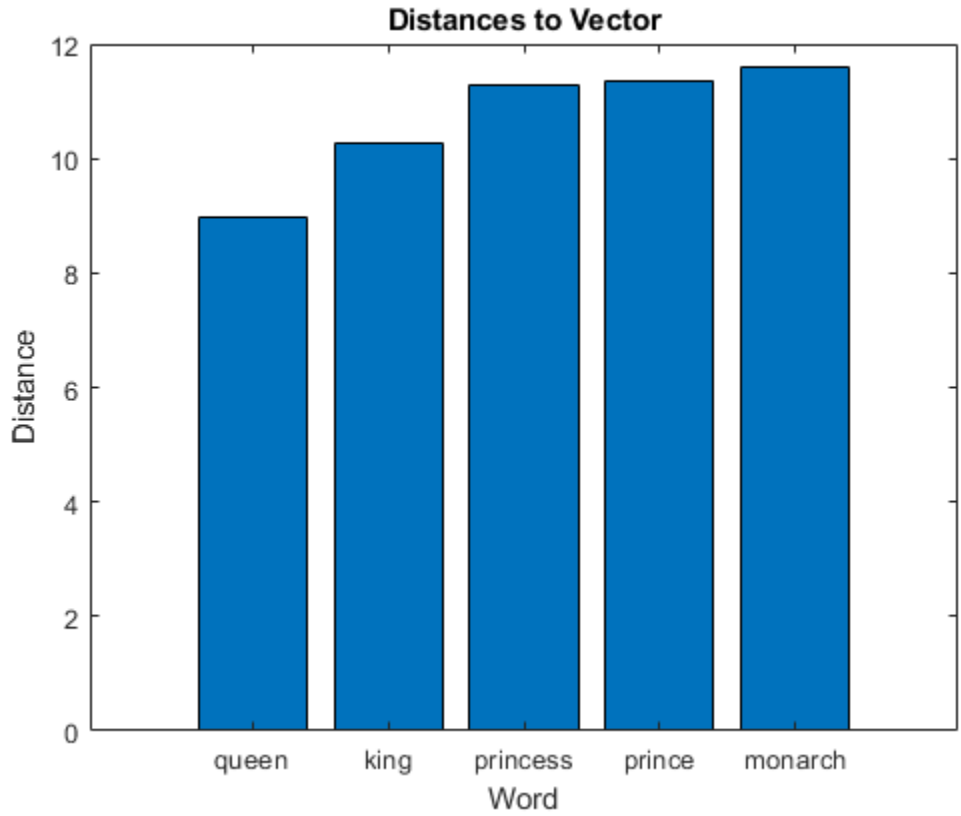
```
king = word2vec(emb,"king");
man = word2vec(emb,"man");
woman = word2vec(emb,"woman");
```

Map the vector `king - man + woman` to a word using `vec2word`. Find the top five closest words using the Euclidean distance metric.

```
k = 5;
M = king - man + woman;
[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
word embedding

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.

`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

`fastTextWordEmbedding` | `ismember` | `readWordEmbedding` | `trainWordEmbedding` | `word2vec` | `wordEmbedding` | `writeWordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

word2vec

Map word to embedding vector

Syntax

```
M = word2vec(emb,words)
```

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.

Examples

Explore Word Embedding

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]
```

Map the words "king", "man", and "woman" to vectors using `word2vec`.

```
king = word2vec(emb,"king");
man = word2vec(emb,"man");
woman = word2vec(emb,"woman");
```

Map the vector `king - man + woman` to a word using `vec2word`.

```
word = vec2word(emb,king - man + woman)
```

```
word =
"queen"
```

Input Arguments

`emb` — Input word embedding

word embedding

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. Each row of `M` corresponds to a word embedding vector for the corresponding entry in `words`. `M` has `emb.Dimension` columns.

See Also

`fastTextWordEmbedding` | `ismember` | `readWordEmbedding` | `trainWordEmbedding` | `vec2word` | `wordEmbedding` | `writeWordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”

“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

wordcloud

Create word cloud chart from text, bag-of-words model, bag-of-n-grams model, or LDA model

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, see `wordcloud` (MATLAB).

Syntax

```
wc = wordcloud(str)

wc = wordcloud(tbl,wordVar,sizeVar)
wc = wordcloud(words,sizeData)
wc = wordcloud(C)

wc = wordcloud(bag)
wc = wordcloud(ldaMdl,topicIdx)

wc = wordcloud(parent, ___ )

wc = wordcloud( ___ ,Name,Value)
```

Description

`wc = 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.

`wc = 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.

`wc = wordcloud(words,sizeData)` creates a word cloud chart from elements of `words` with word sizes specified by `SizeData`.

`wc = wordcloud(C)` creates a word cloud chart from the elements of categorical array `C` using frequency counts.

`wc = wordcloud(bag)` creates a word cloud chart from the bag-of-words or bag-of-n-grams model `bag`.

`wc = wordcloud(ldaMdl,topicIdx)` creates a word cloud chart from the topic with index `topicIdx` of the LDA model `ldaMdl`.

`wc = wordcloud(parent, ___ )` creates the word cloud in the figure, panel, or tab specified by `parent`.

`wc = wordcloud( ___ ,Name,Value)` specifies additional `WordCloudChart` properties using one or more name-value pair arguments.

Examples

Create Word Cloud from Text Data

Extract the text from `sonnets.txt` using `extractFileText`.

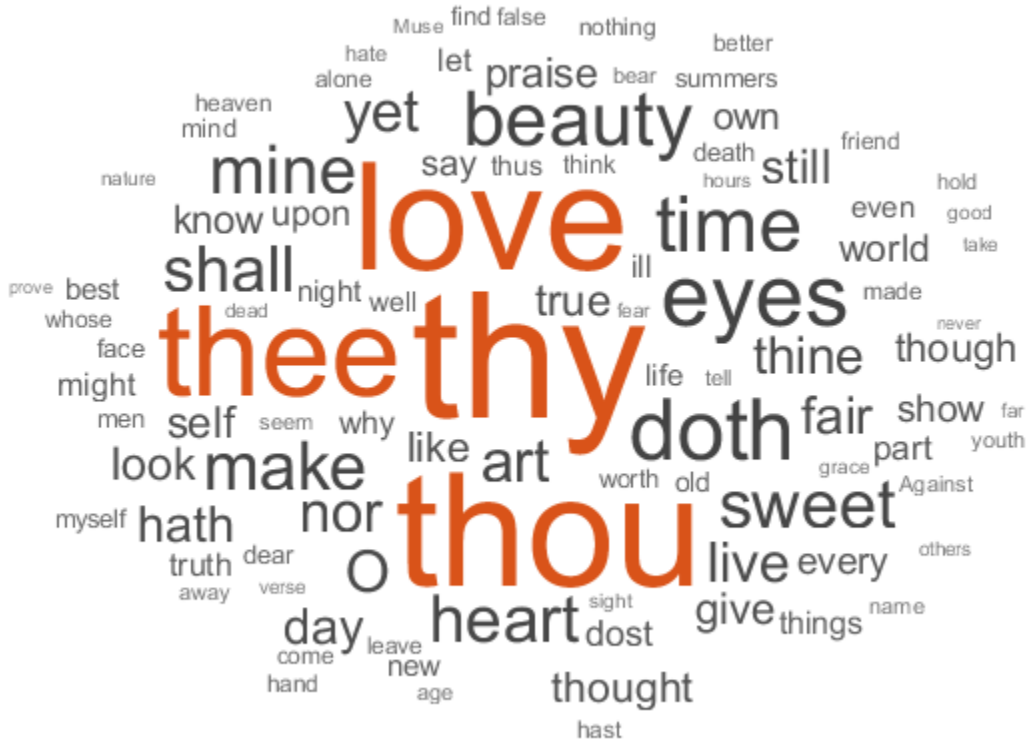
```
str = extractFileText("sonnets.txt");
extractBefore(str,"II")
```

```
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)
```




```
ans =
  WordCloudChart with properties:

          WordData: [1x2659 string]
          SizeData: [1x2659 double]
    MaxDisplayWords: 100

  Show all properties
```

Create Word Cloud from Table

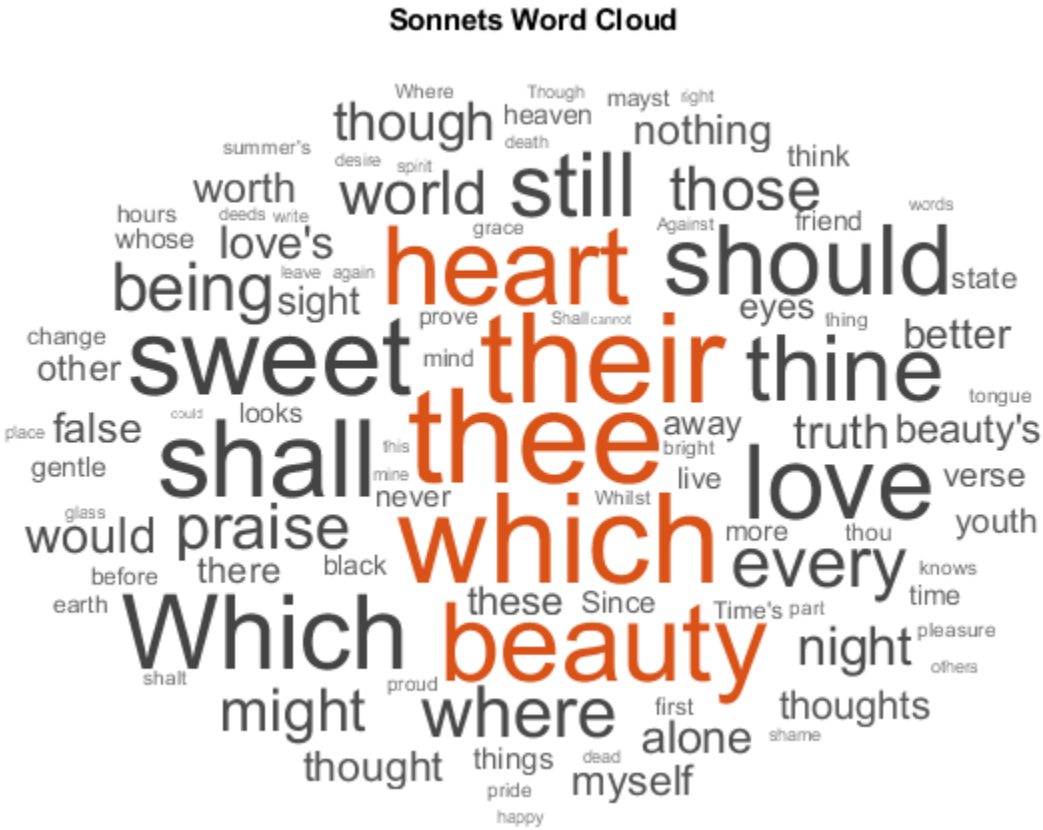
Load the example data `sonnetsTable`. The table `tbl` contains a list of words in the variable `Word`, and the corresponding frequency counts in the variable `Count`.

```
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 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);
```

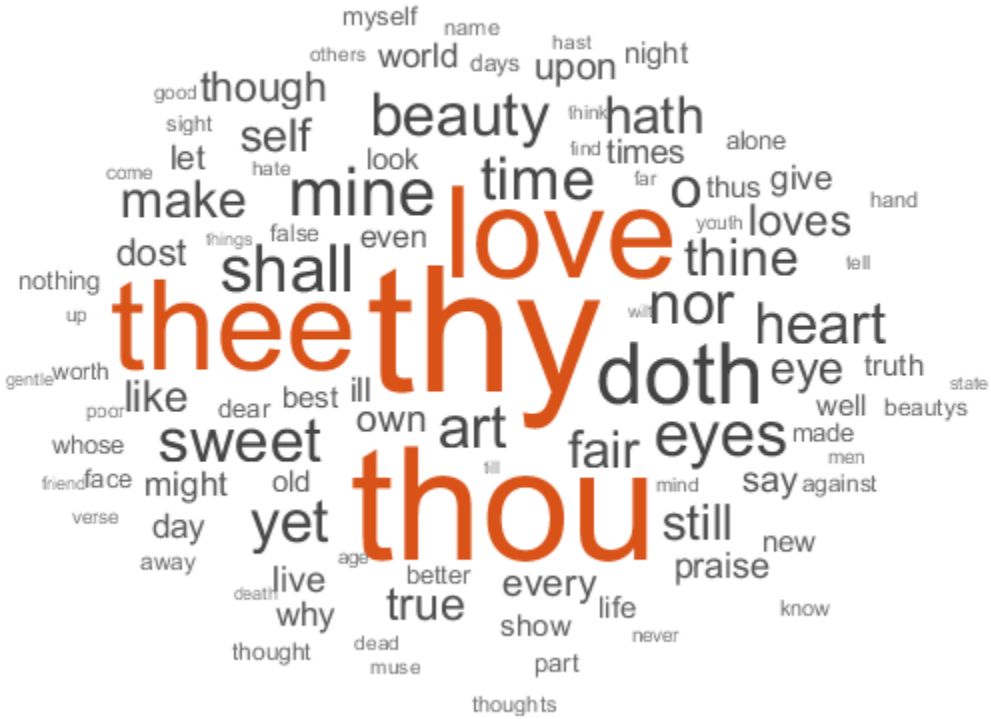
Create a bag-of-words model using `bagOfWords`.

```
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 LDA Topic

Load the `sonnetsDocuments` data and create a bag-of-words model. `sonnetsDocuments` returns a `tokenizedDocument` array of preprocessed versions of Shakespeare's sonnets.

To use the example file `sonnetsDocuments.m`, add the example folder to the path. To reproduce the results, set `rng` to `'default'`.

```
exampleFolder = genpath(fullfile(matlabroot,'examples','textanalytics'));
addpath(exampleFolder)
rng('default')
```

Load the `sonnetsDocuments` data and create a bag-of-words model.

```
documents = sonnetsDocuments;
bag = bagOfWords(documents)
```

```
bag =
  bagOfWords with properties:

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

Fit an LDA model with 20 topics.

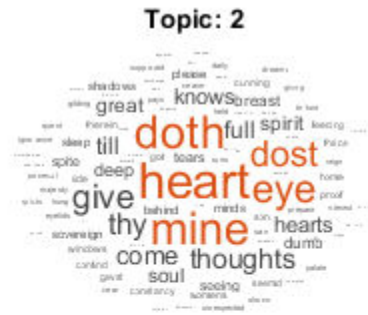
```
mdl = fitlda(bag,20)
```

```
Initial topic assignments sampled in 0.0410732 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 |       0.13 | 5.4884e-02 |  8.028e+02 |         5.000 |             0 |
|          2 |       0.07 | 4.7400e-03 |  7.778e+02 |         5.000 |             0 |
|          3 |       0.08 | 3.4597e-03 |  7.602e+02 |         5.000 |             0 |
|          4 |       0.08 | 3.4662e-03 |  7.430e+02 |         5.000 |             0 |
|          5 |       0.07 | 2.9259e-03 |  7.288e+02 |         5.000 |             0 |
|          6 |       0.08 | 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]
                   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
```



Remove the example folder from the path using `rmpath`.

```
rmpath(exampleFolder)
```

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`

`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 the n-grams as individual words.

`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 single 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`.

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]`. Alternatively, you can specify some common colors by name. This table lists the long and short color name options and the equivalent RGB triplet values.

Option	Description	Equivalent RGB Triplet
`'red'` or `'r'`	Red	`[1 0 0]`
`'green'` or `'g'`	Green	`[0 1 0]`
`'blue'` or `'b'`	Blue	`[0 0 1]`
`'yellow'` or `'y'`	Yellow	`[1 1 0]`
`'magenta'` or `'m'`	Magenta	`[1 0 1]`
`'cyan'` or `'c'`	Cyan	`[0 1 1]`

Option	Description	Equivalent RGB Triplet
`'white'` or `'w'`	White	`[1 1 1]`
`'black'` or `'k'`	Black	`[0 0 0]`

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.

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]`. Alternatively, you can specify some common colors by name. This table lists the long and short color name options and the equivalent RGB triplet values.

Option	Description	Equivalent RGB Triplet
`'red'` or `'r'`	Red	`[1 0 0]`
`'green'` or `'g'`	Green	`[0 1 0]`
`'blue'` or `'b'`	Blue	`[0 0 1]`
`'yellow'` or `'y'`	Yellow	`[1 1 0]`
`'magenta'` or `'m'`	Magenta	`[1 0 1]`
`'cyan'` or `'c'`	Cyan	`[0 1 1]`
`'white'` or `'w'`	White	`[1 1 1]`
`'black'` or `'k'`	Black	`[0 0 0]`

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.

See Also

`bagOfNgrams` | `bagOfWords` | `textscatter` | `textscatter3` | `wordCloudCounts`

Topics

“Visualize Text Data Using Word Clouds”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Analyze Text Data Using Topic Models”

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.

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"       234
    "love"       215
    "thee"       161
    "eyes"        93
    "doth"        88
    "time"        78
    "beauty"      75
```

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

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".

See Also

`textscatter` | `textscatter3` | `wordcloud`

Topics

“Visualize Text Data Using Word Clouds”
“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Analyze Text Data Using Topic Models”

Introduced in R2017b

wordEmbedding

Map words to vectors and back

Description

Word embeddings, popularized by the word2vec, GloVe, and fastText libraries, map 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
nonnegative integer

Dimension of the word embedding, specified as a nonnegative 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

ismember	Test word is member of word embedding
vec2word	Map embedding vector to word
word2vec	Map word to embedding vector
writeWordEmbedding	Write word embedding file

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"
```

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: 0        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]
```

Explore Word Embedding

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]
```

Map the words "king", "man", and "woman" to vectors using `word2vec`.

```
king = word2vec(emb,"king");
man = word2vec(emb,"man");
woman = word2vec(emb,"woman");
```

Map the vector `king - man + woman` to a word using `vec2word`.

```
word = vec2word(emb,king - man + woman)
```

```
word =
"queen"
```

Find Closest Words to Vector

Find the top five closest words to a word embedding vector and their distances.

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]
```

Map the words "king", "man", and "woman" to vectors using word2vec.

```
king = word2vec(emb,"king");
man = word2vec(emb,"man");
woman = word2vec(emb,"woman");
```

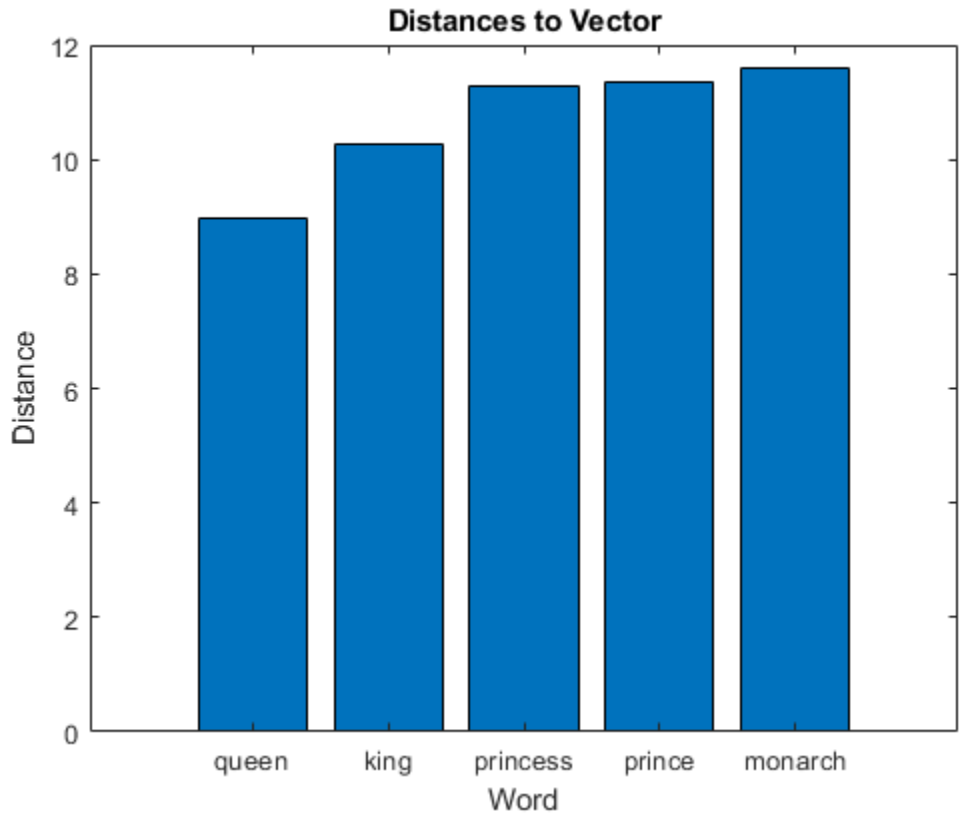
Map the vector `king - man + woman` to a word using `vec2word`. Find the top five closest words using the Euclidean distance metric.

```
k = 5;
M = king - man + woman;
```

```
[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")
```



- “Visualize Word Embeddings Using Text Scatter Plots”

- “Prepare Text Data for Analysis”
- “Extract Text Data from Files”

See Also

`fastTextWordEmbedding` | `ismember` | `readWordEmbedding` | `textscatter` | `textscatter3` | `trainWordEmbedding` | `vec2word` | `word2vec` | `writeWordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b

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:

(1,1) 6 tokens: an example of a short sentence
(2,1) 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:

(1,1) 6 tokens: an example of a short sentence
(2,1) 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` | `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: 0        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

word embedding

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

`fastTextWordEmbedding` | `ismember` | `readWordEmbedding` | `trainWordEmbedding` | `vec2word` | `word2vec` | `wordEmbedding`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”
“Prepare Text Data for Analysis”
“Extract Text Data from Files”

Introduced in R2017b