

Text Analytics Toolbox™

Reference



MATLAB®

R2017b

 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

© COPYRIGHT 2017 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

September 2017 Online Only

New for Version 1.0

1	Functions — Alphabetical List
----------	--------------------------------------

Functions — Alphabetical List

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 of the words of a single document.

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 — Words in model

string vector

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

<code>encode</code>	Encode documents as matrix of word counts
<code>tfidf</code>	Term Frequency–Inverse Document Frequency (tf-idf) matrix
<code>topkeywords</code>	Most important words in bag-of-words model or LDA topic
<code>addDocument</code>	Add documents to bag-of-words model
<code>removeDocument</code>	Remove documents from bag-of-words model
<code>removeEmptyDocuments</code>	Remove empty documents from tokenized document array or bag-of-words model
<code>removeWords</code>	Remove selected words from document or bag-of-words model
<code>removeInfrequentWords</code>	Remove words with low counts from bag-of-words model

Examples

Create Bag-of-Words Model

Load the `sonnetsDocuments` data. `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;
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 issuel
(10,1) 69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt
```

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

```
bag =
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	
0	0	0	0	0	
...					

View the top 10 words and their total counts.

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

```
T=10x2 table null
  Word      Count
-----
"thy"      281
"thou"     234
"love"     162
"thee"     161
"doth"     88
"mine"     63
"shall"    59
"eyes"     56
"sweet"    55
"time"     53
```

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

```
rmpath(exampleFolder)
```

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 7 words and 4 documents:
```

	a	an	another	example	final	...
1	1	2	0	1	0	
0	0	0	3	1	0	
...						

Import Text Data Using a File Datastore

Import three example sonnet text files using a file datastore.

Create a file datastore for the example sonnet text files. The example sonnets have filenames "exampleSonnet1.txt", "exampleSonnet2.txt", and "exampleSonnet3.txt". Specify the read function to be `extractFileText`.

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

```
fds =
FileDatastore with properties:
```

```
Files: {
    ' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet1.t
    ' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet2.t
    ' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet3.t
}
```

```
UniformRead: 0
  ReadFcn: @extractFileText
```

Create an empty bag-of-words model.

```
bag = bagOfWords
```

```
bag =
Empty bagOfWords
```

Loop over the files in the datastore and read each file. Tokenize the text 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 224 words and 3 documents:
```

From	fairest	creatures	we	desire	...
1	1	1	1	1	
0	0	0	0	0	
...					

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)
```

```
bag =  
bagOfWords with 7 words and 2 documents:
```

an	example	of	a	short	...
1	1	1	1	1	1
0	0	0	1	1	

Remove the stop words using the `stopWords` function to input a list of stop words directly into `removeWords`.

```
newBag = removeWords(bag, stopWords)
```

```
newBag =  
bagOfWords with 4 words and 2 documents:
```

example	short	sentence	second
1	1	1	0
0	1	1	1

Most Frequent Words of Bag-of-Words Model

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

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.

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

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

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

```
bag =  
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	

```

0      0      0      0      0
...

```

Find the top five words.

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

```
T=5x2 table
```

Word	Count
"thy"	281
"thou"	234
"love"	162
"thee"	161
"doth"	88

Find the top 20 words.

```
k = 20;
```

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

```
T=20x2 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

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

```
rmpath(exampleFolder)
```

Create Tf-idf Matrix

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

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.

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

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

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

```
bag =  
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	
0	0	0	0	0	
...					

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

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

```
ans =
```

```
Columns 1 through 7
```

3.6507	4.3438	2.7344	3.6507	4.3438	2.2644	3.2452
0	0	0	0	0	4.5287	0
0	0	0	0	0	0	0
0	0	0	0	0	2.2644	0

```

0          0          0          0          0          2.2644          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          2.2644          0
0          0          2.7344          0          0          0          0

```

Columns 8 through 10

```

3.8918      2.4720      2.5520
0           0           0
0           0          2.5520
0           0           0
0           0           0
0           0           0
0           0           0
0           0           0
0           0          2.5520
0           0           0

```

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

```
rmpath(exampleFolder)
```

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

See Also

`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.

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 parameter, specified as a positive scalar.

WordConcentration — Word concentration

1 (default) | nonnegative scalar

Word concentration parameter, 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.

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.

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.

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

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

<code>logp</code>	Document log-probabilities and goodness of fit of LDA model
<code>predict</code>	Predict top LDA topics of documents
<code>resume</code>	Resume fitting LDA model
<code>topkwords</code>	Most important words in bag-of-words model or LDA topic
<code>transform</code>	Transform documents into lower-dimensional space
<code>wordcloud</code>	Create word cloud chart from bag-of-words model or LDA model

Examples

Fit LDA Model

Fit a Latent Dirichlet Allocation model to a collection of documents.

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

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
           1           1           1           1           1
           0           0           0           0           0
           ...
```

Fit an LDA model with 20 topics.

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

```
Initial topic assignments sampled in 0.0286443 seconds.
```

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|           0 | 0.01 | Inf | 1.159e+03 | 5.000 | 0 |
|           1 | 0.03 | 5.4884e-02 | 8.028e+02 | 5.000 | 0 |
|           2 | 0.03 | 4.7400e-03 | 7.778e+02 | 5.000 | 0 |
|           3 | 0.03 | 3.4597e-03 | 7.602e+02 | 5.000 | 0 |
|           4 | 0.03 | 3.4662e-03 | 7.430e+02 | 5.000 | 0 |
|           5 | 0.03 | 2.9259e-03 | 7.288e+02 | 5.000 | 0 |
|           6 | 0.03 | 6.4180e-05 | 7.291e+02 | 5.000 | 0 |
=====
```

```
mdl =
ldaModel with properties:

    NumTopics: 20
 WordConcentration: 1
 TopicConcentration: 5
 CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
 DocumentTopicProbabilities: [154x20 double]
 TopicWordProbabilities: [3092x20 double]
```



```
rmpath(exampleFolder)
```

Highest Probability Words of LDA Topic

Create a table of the words with highest probability of an 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);
```

Fit an LDA model with 20 topics.

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

Initial topic assignments sampled in 0.0249502 seconds.

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
|=====|=====|=====|=====|=====|=====|
|         0 |      0.00 |          Inf | 1.159e+03 | 5.000 |          0 |
|         1 |      0.03 | 5.4884e-02 | 8.028e+02 | 5.000 |          0 |
|         2 |      0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |          0 |
|         3 |      0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |          0 |
|         4 |      0.03 | 3.4662e-03 | 7.430e+02 | 5.000 |          0 |
|         5 |      0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |          0 |
|         6 |      0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |          0 |
|=====|=====|=====|=====|=====|=====|
```

```
mdl =
  ldaModel with properties:
```

```
        NumTopics: 20
        WordConcentration: 1
        TopicConcentration: 5
        CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
        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)
```

Find the top 20 words of the first topic and use inverse mean scaling on the scores.

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

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

```
rmpath(exampleFolder)
```

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.

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

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 3092 words and 154 documents:
```

```

    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
    ...

```

Fit an LDA model with 20 topics.

```

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

```

Initial topic assignments sampled in 0.0253534 seconds.

```

=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|         0 |    0.01 |      Inf | 1.159e+03 | 5.000 |         0 |
|         1 |    0.04 | 5.4884e-02 | 8.028e+02 | 5.000 |         0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |         0 |
|         3 |    0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |         0 |
|         4 |    0.02 | 3.4662e-03 | 7.430e+02 | 5.000 |         0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |         0 |
|         6 |    0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |         0 |
=====

```

```

mdl =
    ldaModel with properties:
        NumTopics: 20
        WordConcentration: 1
        TopicConcentration: 5
        CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
        DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
        Vocabulary: [1x3092 string]
        FitInfo: [1x1 struct]

```

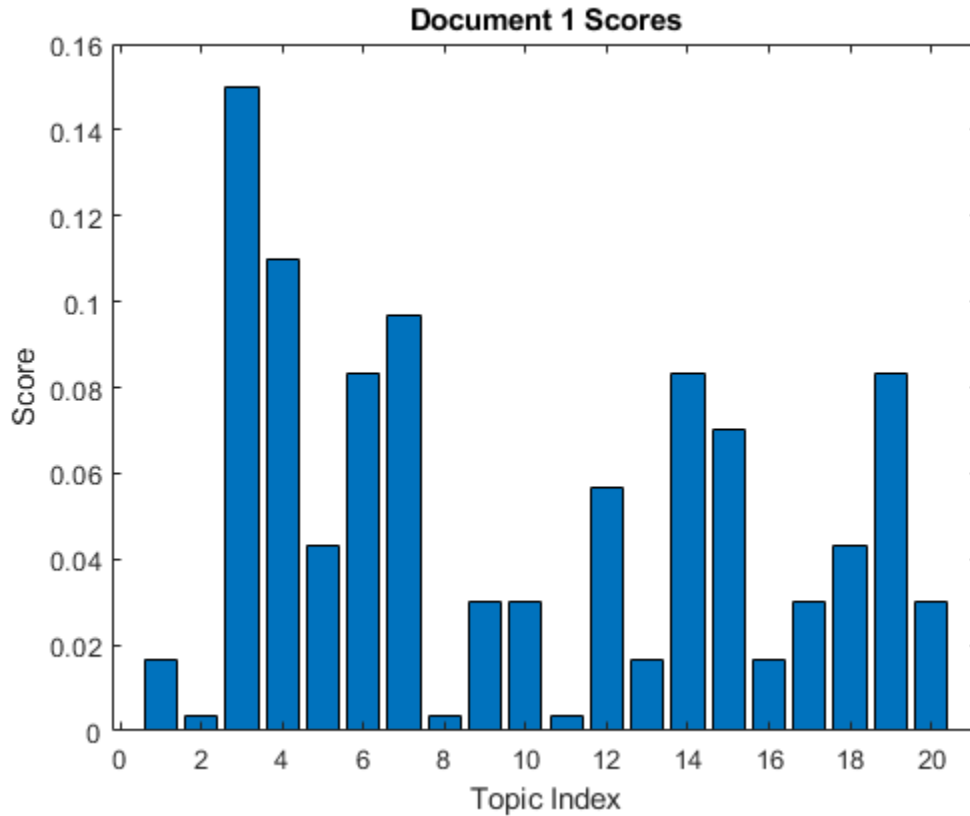
View the topic probabilities of the first document in the training data.

```

topicMixtures = mdl.DocumentTopicProbabilities;
figure
bar(topicMixture(1,:))

```

```
title("Document 1 Topic Probabilities")
xlabel("Topic Index")
ylabel("Probability")
```



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

```
rmpath(exampleFolder)
```

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

See Also

`bagOfWords` | `lsaModel`

Topics

“Analyze Text Data Using Topic Models”

“Prepare Text Data for Analysis”

“Extract Text Data From Files”

Introduced in R2017b

IsaModel

Latent semantic analysis (LSA) model

Description

A latent semantic analysis (LSA) model discovers relationships between documents and the words that they contain.

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 — Words in model

string vector

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 `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.

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

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

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

```
bag =  
bagOfWords with 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...  
         1         1         1         1         1  
         0         0         0         0         0  
         ...
```

Fit an LSA model with 20 components.

```
numComponents = 20;  
mdl = fitlsa(bag,numComponents)
```

```
mdl =  
lsaModel with properties:
```

```
    NumComponents: 20  
    ComponentWeights: [2.7866e+03 515.5889 443.6428 316.4191 295.4065 261.8927 2  
    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 =

    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

```

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

```
rmpath(exampleFolder)
```

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: ["I" "enjoy" "ham" ", " "eggs" "and" "bac
        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)
```

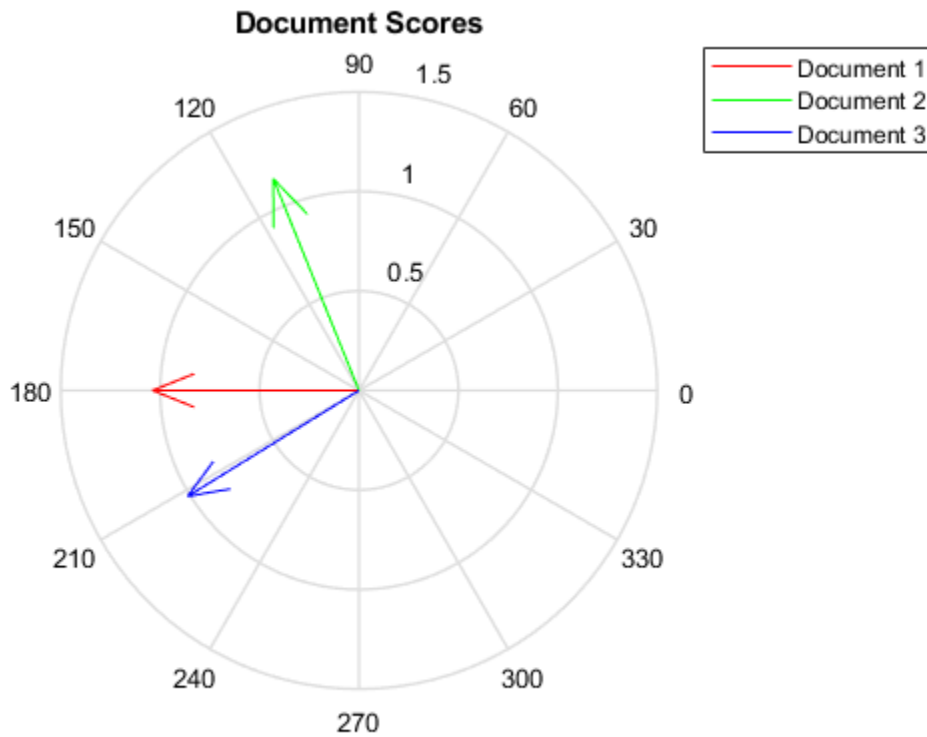
```
D =
```

```

         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

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.

Creation

Syntax

```
documents = tokenizedDocument
documents = tokenizedDocument(str)
documents = tokenizedDocument(C, 'TokenizeMethod', 'none')
```

Description

`documents = tokenizedDocument` creates an empty array of tokenized documents.

`documents = tokenizedDocument(str)` tokenizes the elements of `str` into words and returns an array of tokenized documents.

The size of `documents` is the same as the size of `str`.

`documents = tokenizedDocument(C, 'TokenizeMethod', 'none')` converts the cell array `C`, where the elements of `C` are split into words, to an array of tokenized documents.

The size of `documents` is the same as the size of `C`.

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`

C — Pretokenized documents

cell array of string arrays | cell array of character vectors

Pretokenized documents, specified as a cell array of string arrays, or a cell array of character vectors.

If your input text has already been split into words, then specify the `TokenizeMethod` option to be `'none'`. The function returns an array of tokenized documents, with words given by the elements of `C`. The elements of `C` must be row vectors of strings, or `C` must be a row of character vectors. If `C` is a row of character vectors, then `tokenizedDocument` returns a scalar tokenized document.

Example: `{"an", "example", "document"}; {"another", "document"}`

Example: `{'an', 'example', 'document'}`

If you want to tokenize the input text, then use the syntax `documents = tokenizedDocuments(str)`, where `str` is a string array, character vector, or cell array of character vectors.

Data Types: `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: `'TokenizeMethod', 'none'` specifies to not tokenize the input text.

TokenizeMethod — Method to tokenize documents

`'unicode'` (default) | `'none'`

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

- 'unicode' – Tokenize input text into words. The elements of `C` must be scalar strings, or character vectors.
- 'none' – Do not tokenize the input text. The elements of `C` must be row vectors of strings, or `C` must be a row of character vectors.

Example: `'TokenizeMethod', 'none'`

Properties

Vocabulary — Unique words in documents

string array

Unique words in documents, specified as a string array. The words do not appear in any particular order.

Data Types: `string`

Object Functions

Conversion

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

Exploration

<code>context</code>	Search documents for word occurrences in context
<code>doclength</code>	Length of documents in document array

Manipulation

<code>docfun</code>	Apply function to words in documents
<code>erasePunctuation</code>	Erase punctuation from text and documents
<code>regexprep</code>	Replace text in words of documents using regular expression
<code>replace</code>	Find and replace substrings in documents

<code>removeEmptyDocuments</code>	Remove empty documents from tokenized document array or bag-of-words model
<code>removeWords</code>	Remove selected words from document or bag-of-words model
<code>normalizeWords</code>	Remove inflections from words using the Porter stemmer
<code>lower</code>	Convert documents to lowercase
<code>upper</code>	Convert documents to uppercase
<code>plus</code>	Append documents

Export

`writeTextDocument` Write documents to text file

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
```

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"])

documents =
  2×1 tokenizedDocument:

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

Remove the stop words using the `stopWords` function to input a list of stop words directly into `removeWords`.

```
newDocuments = removeWords(documents, stopWords)

newDocuments =
  2×1 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"])

documents =
  2×1 tokenizedDocument:

(1,1) 6 tokens: a strongly worded collection of words
(2,1) 4 tokens: 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 `sonnetsDocuments` data. `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.

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

Load the `sonnetsDocuments` data.

```
documents = sonnetsDocuments;
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 issuel
(10,1) 69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt
```

Search for the word "life".

```
T = context(documents, "life")
```

```
T =
```

```
23x3 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
"e rehearse let love even life decay lest wise world lo"	71	46
"st bail shall carry away life hath line interest memor"	74	8
"art thou hast lost dregs life prey worms body dead cow"	74	36
" thoughts food life sweetseasond showers gro"	75	3
"tten name hence immortal life shall though once gone w"	81	18
" beauty mute others give life bring tomb lives life fa"	83	45
"ve life bring tomb lives life fair eyes poets praise d"	83	49
" steal thyself away term life thou art assured mine li"	92	7
"fe thou art assured mine life longer thy love stay dep"	92	12
" fear worst wrongs least life hath end better state be"	92	26
"anst vex inconstant mind life thy revolt doth lie o ha"	92	41
" fame faster time wastes life thou preventst scythe cr"	100	69
"ess harmful deeds better life provide public means pub"	111	10
"ate hate away threw savd life saying "	145	51
" many nymphs vovd chaste life keep came tripping maide"	154	15

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

```
rmpath(exampleFolder)
```

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

See Also

`bagOfWords`

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

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 either 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 — Words in model

string vector

Words in the model, specified as a string vector.

Data Types: `string`

Object Functions

<code>ismember</code>	Test word is member of word embedding
<code>vec2word</code>	Map embedding vector to word
<code>word2vec</code>	Map word to embedding vector
<code>writeWordEmbedding</code>	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.

Train a word embedding using the example data returned by `sonnetsDocuments`. This function returns preprocessed versions of Shakespeare's sonnets as a `tokenizedDocument` array.

To use the example file `sonnetsDocuments.m`, add the example folder to the path.

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

Load the `sonnetsDocuments` data.

```
documents = sonnetsDocuments;  
emb = trainWordEmbedding(documents)
```

```
Training: 100% Loss: 2.78156 Remaining time: 0 hours 0 minutes.
```

```
emb =  
wordEmbedding with properties:  
  
    Dimension: 100  
    Vocabulary: [1×401 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: [1×401 string]
```

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

```
rmpath(exampleFolder)
```

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`.

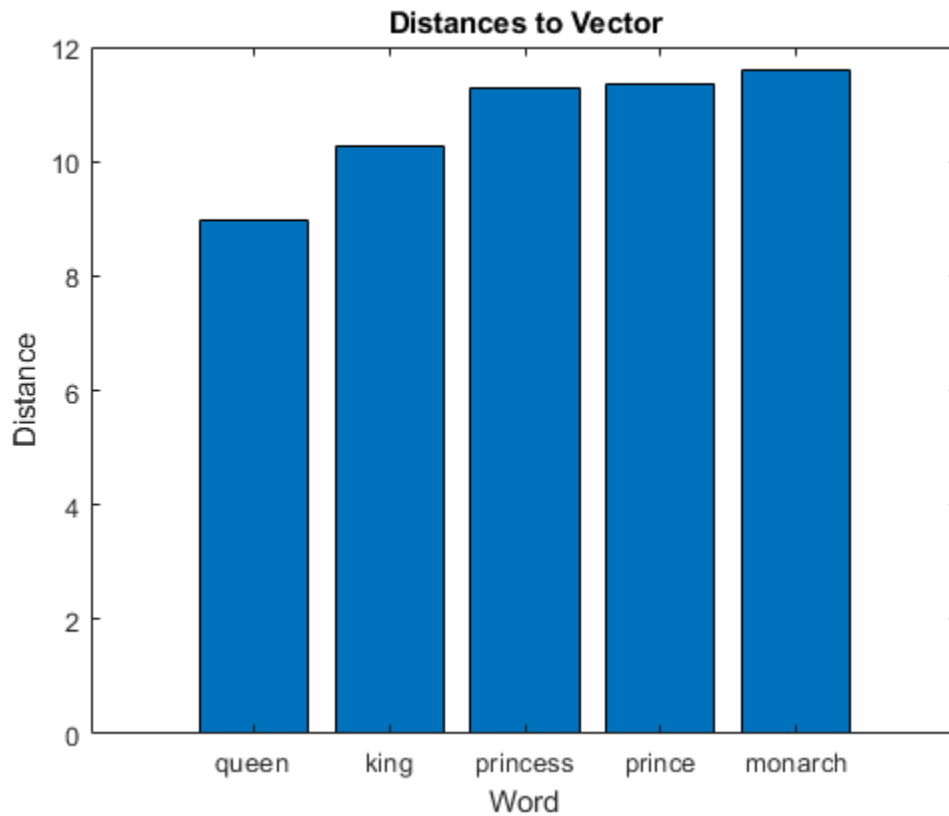
```
king = word2vec(emb, "king");
man = word2vec(emb, "man");
woman = word2vec(emb, "woman");
```

Map the vector $\text{king} - \text{man} + \text{woman}$ 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

`textscatter` | `textscatter3`

Topics

“Visualize Word Embeddings Using Text Scatter Plots”

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

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 by 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 the 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.

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 the 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 object. MATLAB executes the callback before destroying the object so that the callback can access its property values.

If you specify this callback using a function handle, then MATLAB passes two arguments to the callback function when executing the callback:

- Deleted object — You can access properties of the object from within the callback function. You also can access the object through the `CallbackObject` property of the root, which can be queried using the `gcbo` function.
- Event data — This argument is empty for this property. Replace it with the tilde character (~) in the function definition to indicate that this argument is not used.

For more information on how to use function handles to define callback functions, see “Callback Definition” (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

addDocument

Add documents to bag-of-words model

Syntax

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

Description

`newBag = addDocument (bag, documents)` adds documents to the bag-of-words 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 7 words and 2 documents:
```

an	example	of	a	short	...
1	1	1	1	1	
0	0	0	1	1	

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 9 words and 4 documents:  
  
    an   example   of   a   short   ...  
    1     1     1   1   1     1  
    0     0     0   1   1     1  
    ...
```

Import Text Data Using a File Datastore

Import three example sonnet text files using a file datastore.

Create a file datastore for the example sonnet text files. The example sonnets have filenames "exampleSonnet1.txt", "exampleSonnet2.txt", and "exampleSonnet3.txt". Specify the read function to be `extractFileText`.

```
fds = fileDatastore('exampleSonnet*.txt', 'ReadFcn', @extractFileText)  
  
fds =  
FileDatastore with properties:  
  
    Files: {  
        ' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet1.t  
        ' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet2.t  
        ' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet3.t  
    }  
    UniformRead: 0  
    ReadFcn: @extractFileText
```

Create an empty bag-of-words model.

```
bag = bagOfWords  
  
bag =  
Empty bagOfWords
```

Loop over the files in the datastore and read each file. Tokenize the text 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 224 words and 3 documents:
```

From	fairest	creatures	we	desire	...
1	1	1	1	1	
0	0	0	0	0	
...					

Input Arguments

bag — Input bag-of-words model

bagOfWords object

Input bag-of-words model, specified as a bagOfWords 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 of the words of a single document.

Output Arguments

newBag — Output bag-of-words model

bagOfWords object

Output bag-of-words model, returned as a bagOfWords object.

See Also

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

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

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 `sonnetsDocuments` data. `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.

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

Load the `sonnetsDocuments` data.

```
documents = sonnetsDocuments;  
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 issuel  
(10,1) 69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt
```

Search for the word "life".

```
T = context(documents, "life")
```

```
T =
```

```
23x3 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
"e rehearse let love even life decay lest wise world lo"	71	46
"st bail shall carry away life hath line interest memor"	74	8
"art thou hast lost dregs life prey worms body dead cow"	74	36
" thoughts food life sweetseasond showers gro"	75	3
"tten name hence immortal life shall though once gone w"	81	18

" beauty mute others give life bring tomb lives life fa"	83	45
"ve life bring tomb lives life fair eyes poets praise d"	83	49
" steal thyself away term life thou art assured mine li"	92	7
"fe thou art assured mine life longer thy love stay dep"	92	12
" fear worst wrongs least life hath end better state be"	92	26
"anst vex inconstant mind life thy revolt doth lie o ha"	92	41
" fame faster time wastes life thou preventst scythe cr"	100	69
"ess harmful deeds better life provide public means pub"	111	10
"ate hate away threw savd life saying "	145	51
" many nymphs vovd chaste life keep came tripping maide"	154	15

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

```
rmpath(exampleFolder)
```

Specify Context Length

Load the `sonnetsDocuments` data. `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.

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

Load the `sonnetsDocuments` data.

```
documents = sonnetsDocuments;
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 issuel
(10,1) 69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt.
```

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

```
T = context(documents, "life", 15)
```

```
T =
```

```
23x3 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
" let love even life decay lest wis"	71	46
"all carry away life hath line inte"	74	8
"ast lost dregs life prey worms bod"	74	36
" thoughts food life sweetseasond s"	75	3
"hence immortal life shall though o"	81	18
"te others give life bring tomb liv"	83	45
"ing tomb lives life fair eyes poet"	83	49
"self away term life thou art assur"	92	7
"t assured mine life longer thy lov"	92	12
"t wrongs least life hath end bette"	92	26
"nconstant mind life thy revolt dot"	92	41
"er time wastes life thou preventst"	100	69
"l deeds better life provide public"	111	10
"way threw savd life saying "	145	51
"hs vovd chaste life keep came trip"	154	15

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


```
rmpath(exampleFolder)
```

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 decay ; 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

scalar string | character vector | scalar cell array

Word to find in context, specified as a scalar string, 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 = 1×2 string array  
    "<>"    "R&D"
```

Replace HTML numeric character references with their unicode equivalent. Unicode character with hex code ` ` 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, 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 words 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 =

     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. The words of `newDocuments(i)` are the output of `func(string(documents(i)))`.

The `docfun` function 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.

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` |
`tokenizedDocument` | `upper`

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

Introduced in R2017b

encode

Encode documents as matrix of word counts

Use `encode` to encode an array of tokenized documents as a matrix of word counts according to a bag-of-words 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 bag-of-words 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( ____, 'DocumentsIn', DocumentsIn)
```

Description

`counts = encode(bag, documents)` returns a matrix of word counts for `documents` based on the bag-of-words model `bag`.

`counts = encode(bag, words)` returns a matrix of word counts for the elements of `words`.

`counts = encode(____, 'DocumentsIn', DocumentsIn)` specifies the orientation of the output word counts.

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 7 words and 2 documents:

```

```

    an  example  of  a  short  ...
    1      1    1  1    1      1
    0      0    0  1    1      1

```

```

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 =

    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 7 words and 2 documents:
```

	an	example	of	a	short	...
1	1	1	1	1	1	
0		0	0	1	1	

```
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 7 words and 2 documents:
```

	an	example	of	a	short	...
1	1	1	1	1	1	
0		0	0	1	1	

```
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 =

     0     0
     0     0
     0     0
     1     1
     0     0
     1     1
     0     1
```

Input Arguments

bag — Input bag-of-words model

`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` 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 of the words of a single document.

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`

DocumentsIn — Orientation of output documents

'rows' (default) | 'columns'

Orientation of output documents in the word count matrix, specified as the comma-separated pair consisting of 'DocumentsIn' and one of the following:

- 'rows' – Return a matrix of word counts with rows corresponding to documents.
- 'columns' – Return a transposed matrix of word counts with columns corresponding to documents.

Data Types: `char`

Output Arguments

counts — Word counts

sparse matrix

Word counts, returned as a sparse matrix of nonnegative integers.

See Also

`bagOfWords` | `tfidf` | `topkeywords`

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, 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.

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")` returns the string `"xy"`. It only removes the tags `<a>` and ``, and does not remove the element `<a>y`.

Examples

Erase HTML and XML Tags and Comments

Erase the tags from some HTML code. The function replaces the `
` 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, 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, character vector, or cell array of character vectors. `str` and `newStr` have the same data type.

See Also

`decodeHTMLEntities` | `erasePunctuation` | `eraseTags` | `tokenizedDocument`

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

Introduced in R2017b

fitlda

Fit latent Dirichlet allocation (LDA) model

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 a bag-of-words model.

`mdl = fitlda(counts,numTopics)` fits an LDA model to the documents represented by a matrix of word counts.

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

Examples

Fit LDA Model

Fit a Latent Dirichlet Allocation model to a collection of documents.

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

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
    ...
```

Fit an LDA model with 20 topics.

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

```
Initial topic assignments sampled in 0.0286443 seconds.
```

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|         0 |    0.01 |      Inf | 1.159e+03 | 5.000 |         0 |
|         1 |    0.03 | 5.4884e-02 | 8.028e+02 | 5.000 |         0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |         0 |
|         3 |    0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |         0 |
|         4 |    0.03 | 3.4662e-03 | 7.430e+02 | 5.000 |         0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |         0 |
|         6 |    0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |         0 |
=====
```

```
mdl =
ldaModel with properties:
```

```
    NumTopics: 20
    WordConcentration: 1
    TopicConcentration: 5
    CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
    DocumentTopicProbabilities: [154x20 double]
    TopicWordProbabilities: [3092x20 double]
    Vocabulary: [1x3092 string]
    FitInfo: [1x1 struct]
```

Visualize the first four topics using word clouds.

Fit LDA Model to 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 =
```

```
          154          3092
```

```
size(uniqueWords)
```

```
ans =
```

```
          1          3092
```

Fit an LDA model with 20 topics. To train for longer, specify the maximum number of iterations to 200.

```
numTopics = 20;
iterations = 200;
mdl = fitlda(counts,numTopics, ...
    'IterationLimit',iterations)
```

```
Initial topic assignments sampled in 0.0281689 seconds.
```

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|         0 |    0.01 |      Inf | 1.159e+03 |  5.000 |          0 |
|         1 |    0.03 | 5.4884e-02 | 8.028e+02 |  5.000 |          0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 |  5.000 |          0 |
|         3 |    0.03 | 3.4597e-03 | 7.602e+02 |  5.000 |          0 |
|         4 |    0.03 | 3.4662e-03 | 7.430e+02 |  5.000 |          0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 |  5.000 |          0 |
|         6 |    0.03 | 6.4180e-05 | 7.291e+02 |  5.000 |          0 |
=====
```

```
mdl =
  ldaModel with properties:
```

```
        NumTopics: 20
        WordConcentration: 1
        TopicConcentration: 5
        CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
        DocumentTopicProbabilities: [154×20 double]
        TopicWordProbabilities: [3092×20 double]
        Vocabulary: [1×3092 string]
        FitInfo: [1×1 struct]
```

The vocabulary of the LDA model is automatically generated. To visualize a topic using a word cloud, you must specify the words and size data manually. Input the vocabulary and the desired column of the `TopicWordProbabilities` property of the LDA model to `wordcloud`.

Visualize the first topic in a word cloud.

```
wordData = uniqueWords;
topicIdx = 1;
sizeData = mdl.TopicWordProbabilities(:,topicIdx);
figure
wordcloud(wordData,sizeData)
```

```
ans =
    WordCloudChart with properties:

        WordData: [1×3092 string]
        SizeData: [1×3092 double]
        MaxDisplayWords: 100

    Show all properties
```




Input Arguments

bag — Input bag-of-words model

bagOfWords object

Input bag-of-words model, specified as a bagOfWords object.

numTopics — Number of topics

positive integer

Number of topics, specified as a positive integer.

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 appears in the *i*th document. Otherwise, the value `counts(i, j)` corresponds to the number of times the *i*th word 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.

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.

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.

This option only applies if you specify the input documents as a matrix of word counts.

Data Types: char

FitTopicConcentration — Option for fitting topic concentration parameter

true (default) | false

Option for fitting topic concentration parameter, specified as the comma-separated pair consisting of 'FitTopicConcentration' and either true or false. If true, then the function fits the scalar concentration parameter in the Dirichlet prior.

Example: 'FitTopicConcentration', false

Data Types: logical

FitTopicProbabilities — Option for fitting topic probabilities

true (default) | false

Option for fitting topic probabilities, specified as the comma-separated pair consisting of 'FitTopicProbabilities' and either true or false. If true, then the function fits the topic probabilities.

Example: 'FitTopicProbabilities', false

Data Types: logical

InitialTopicConcentration — Initial estimate of topic concentration

numTopics/4 (default) | nonnegative scalar

Initial estimate of 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.

Example: 'InitialTopicConcentration', 25

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

Solver — Solver for optimization

'cgs' (default) | 'avb' | 'cvb0'

Solver for optimization, specified as the comma-separated pair consisting of 'Solver' and one of the following:

- 'cgs' – Use collapsed Gibbs sampling. This solver can be more accurate than approximate variational Bayes at the cost of taking longer to run. The `resume` function does not support models fitted with collapsed Gibbs sampling.
- 'avb' – Use approximate variational Bayes. This solver typically runs quicker than collapsed Gibbs sampling and collapsed variational Bayes, but can be less accurate.
- 'cvb0' – Use collapsed variational Bayes, zeroth order.

The `resume` function does not support collapsed Gibbs sampling.

Example: 'Solver', 'avb'

Data Types: char

ValidationData — Validation data to monitor optimization convergence

[] (default) | bagOfWords 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, 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

WordConcentration — Word concentration parameter

1 (default) | nonnegative scalar

Word concentration parameter, 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.

Output Arguments

mdl — Output LDA model

`ldaModel` object

Output LDA model, returned as an `ldaModel` object.

See Also

`bagOfWords` | `fitlsa` | `ldaModel` | `logp` | `lsaModel` | `predict` | `resume` | `transform` | `wordcloud`

Topics

“Analyze Text Data Using Topic Models”

“Prepare Text Data for Analysis”

“Extract Text Data From Files”

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 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 `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.

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

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

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

bag =
  bagOfWords with 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
    ...
```

Fit an LSA model with 20 components.

```
numComponents = 20;
mdl = fitlsa(bag,numComponents)
```

```
mdl =
  lsaModel with properties:
```

```
    NumComponents: 20
  ComponentWeights: [2.7866e+03 515.5889 443.6428 316.4191 295.4065 261.8927 2
  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 =

    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
```

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

```
rmpath(exampleFolder)
```

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 =

      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: [2.7866e+03 515.5889 443.6428 316.4191 295.4065 261.8927 2
    DocumentScores: [154×20 double]
        WordScores: [3092×20 double]
        Vocabulary: [1×3092 string]
    FeatureStrengthExponent: 4
```

Input Arguments

bag — Input bag-of-words model

bagOfWords object

Input bag-of-words model, specified as a bagOfWords object.

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 appears in the *i*th document. Otherwise, the value `counts(i,j)` corresponds to the number of times the *i*th word 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.

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.

This option only applies if you specify the input documents as a matrix of word counts.

Data Types: char

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

`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

Word Member of Embedding

Test 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 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

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

scalar string | character vector | scalar cell array

Delimiter to join words, specified as a scalar string, 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, 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

logp

Document log-probabilities and goodness of fit of LDA model

Syntax

```
logProb = logp(ldaMdl, documents)
logProb = logp(ldaMdl, counts)
[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, 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

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
    ...
```

Fit an LDA model with 20 topics.

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

Initial topic assignments sampled in 0.0218743 seconds.

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|         0 |    0.01 |      Inf | 1.159e+03 | 5.000 |         0 |
|         1 |    0.03 | 5.4884e-02 | 8.028e+02 | 5.000 |         0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |         0 |
|         3 |    0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |         0 |
|         4 |    0.03 | 3.4662e-03 | 7.430e+02 | 5.000 |         0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |         0 |
|         6 |    0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |         0 |
=====
```

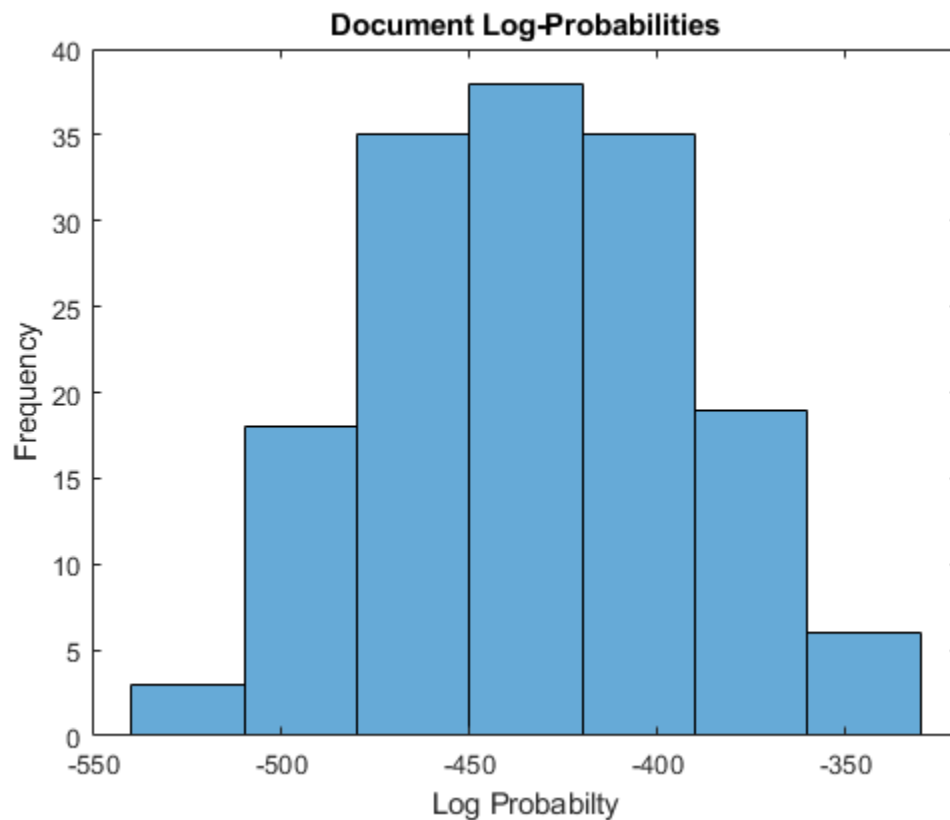
```
mdl =
ldaModel with properties:
```

```
    NumTopics: 20
 WordConcentration: 1
 TopicConcentration: 5
```

```
CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
DocumentTopicProbabilities: [154×20 double]
TopicWordProbabilities: [3092×20 double]
Vocabulary: [1×3092 string]
FitInfo: [1×1 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 Probabilty")
ylabel("Frequency")
title("Document Log-Probabilities")
```



Identify the three documents with the lowest log-probability.

```
[~,idx] = sort(logProbabilities);  
idx(1:3)
```

```
ans =
```

```
146  
19  
65
```

```
documents(idx(1:3))
```

```
ans =  
    3×1 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 brood  
(3,1) 73 tokens: brass nor stone nor earth nor boundless sea sad mortality oersways pow
```

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

```
rmpath(exampleFolder)
```

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 =  
  
    154    3092
```

Fit an LDA model with 20 topics.

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

Initial topic assignments sampled in 0.107692 seconds.

```
=====  
| Iteration | Time per | Relative | Training | Topic | Concentr. |  
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |  
=====  
|           | 0 | 0.18 | Inf | 1.159e+03 | 5.000 | 0 |  
|           | 1 | 0.03 | 5.4884e-02 | 8.028e+02 | 5.000 | 0 |  
|           | 2 | 0.03 | 4.7400e-03 | 7.778e+02 | 5.000 | 0 |  
|           | 3 | 0.03 | 3.4597e-03 | 7.602e+02 | 5.000 | 0 |  
|           | 4 | 0.03 | 3.4662e-03 | 7.430e+02 | 5.000 | 0 |  
|           | 5 | 0.03 | 2.9259e-03 | 7.288e+02 | 5.000 | 0 |  
|           | 6 | 0.03 | 6.4180e-05 | 7.291e+02 | 5.000 | 0 |  
=====
```

```
mdl =
  ldaModel with properties:

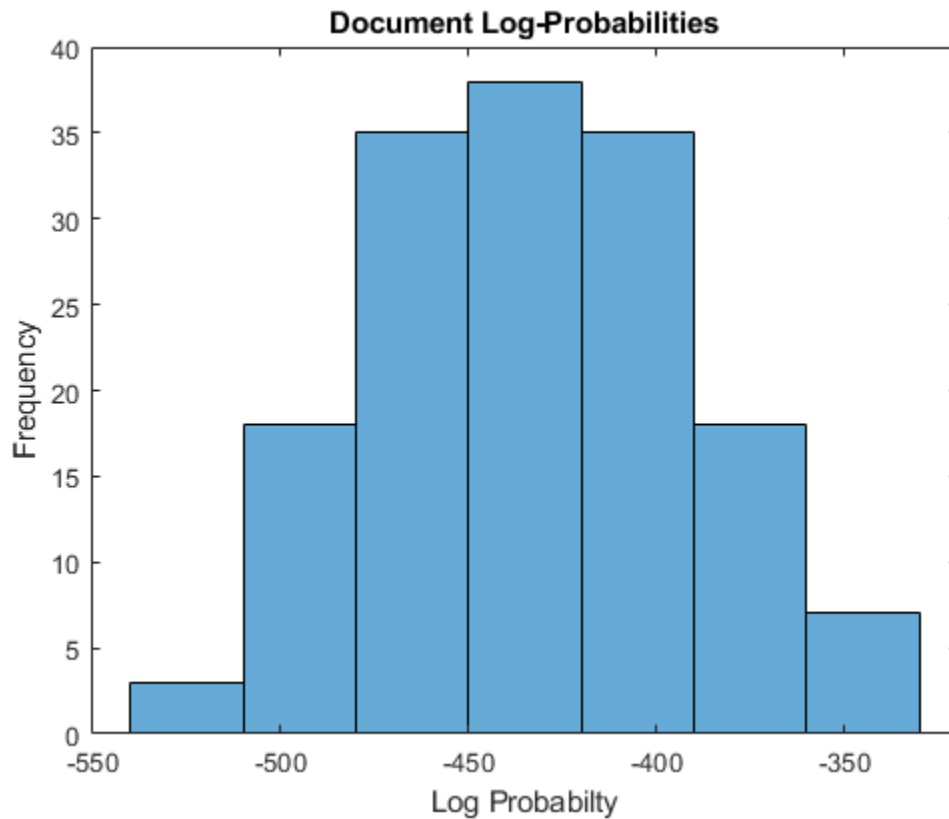
          NumTopics: 20
      WordConcentration: 1
     TopicConcentration: 5
CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
DocumentTopicProbabilities: [154×20 double]
     TopicWordProbabilities: [3092×20 double]
          Vocabulary: [1×3092 string]
           FitInfo: [1×1 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 Probabilty")
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 =
```

```
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.

Load the `sonnetsDocuments` data. `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.

```
documents = sonnetsDocuments;
```

Set aside 10% of the documents at random.

```
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 2909 words and 139 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	
0	0	0	0	0	
...					

Fit an LDA model with 20 topics to the bag-of-words model.

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

```
Initial topic assignments sampled in 0.0181961 seconds.
```

```
=====
```

Iteration	Time per iter., s	Relative Delta log(L)	Training perplexity	Topic concentr.	Concentr. iterations
0	0.00	Inf	1.137e+03	5.000	0
1	0.02	5.5571e-02	7.850e+02	5.000	0
2	0.02	5.2541e-03	7.582e+02	5.000	0
3	0.02	3.9933e-03	7.384e+02	5.000	0
4	0.02	7.2248e-04	7.349e+02	5.000	0
5	0.02	3.8814e-03	7.164e+02	5.000	0
6	0.02	2.1087e-03	7.065e+02	5.000	0
7	0.02	2.0816e-03	6.970e+02	5.000	0
8	0.02	1.9510e-03	6.882e+02	5.000	0
9	0.02	1.4162e-03	6.818e+02	5.000	0
10	0.02	1.4113e-03	6.756e+02	5.000	0
11	0.03	1.0378e-03	6.710e+02	13.255	23
12	0.03	1.1699e-02	7.248e+02	16.536	20
13	0.03	2.5441e-03	7.371e+02	17.269	13
14	0.03	9.7803e-04	7.418e+02	18.155	14
15	0.03	9.8659e-04	7.467e+02	18.650	11
16	0.03	1.9437e-03	7.564e+02	19.555	14
17	0.03	1.0115e-03	7.513e+02	18.746	13
18	0.03	1.3317e-03	7.580e+02	19.781	15
19	0.02	3.1099e-04	7.596e+02	19.851	5
20	0.03	8.9869e-04	7.550e+02	19.181	12
21	0.02	5.1822e-04	7.525e+02	19.364	7
22	0.02	4.5942e-04	7.502e+02	19.278	5
23	0.02	1.0923e-04	7.496e+02	19.250	4
24	0.02	4.2040e-04	7.476e+02	19.079	6
25	0.02	8.8342e-04	7.432e+02	18.516	11
26	0.02	9.9164e-05	7.427e+02	18.606	5

Compute the perplexity of the held-out test set.

```
[~,ppl1] = logp(md11,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 0.0185363 seconds.

Iteration	Time per iter., s	Relative Delta log(L)	Training perplexity	Topic concentr.	Concentr. iterations
0	0.00	Inf	1.068e+03	10.000	0
1	0.05	8.7406e-02	6.095e+02	10.000	0
2	0.04	7.1007e-03	5.826e+02	10.000	0
3	0.04	4.1709e-03	5.674e+02	10.000	0
4	0.04	4.7632e-03	5.506e+02	10.000	0
5	0.04	2.5930e-03	5.417e+02	10.000	0
6	0.04	2.8340e-03	5.321e+02	10.000	0
7	0.04	4.4464e-04	5.336e+02	10.000	0
8	0.04	1.3042e-03	5.293e+02	10.000	0
9	0.04	3.2357e-03	5.187e+02	10.000	0
10	0.04	2.4565e-04	5.179e+02	10.000	0
11	0.05	2.9851e-03	5.083e+02	16.621	17
12	0.05	8.3119e-03	5.356e+02	18.276	13
13	0.05	4.3005e-03	5.503e+02	19.955	13
14	0.04	1.2658e-03	5.547e+02	20.172	6
15	0.05	1.1721e-03	5.589e+02	20.424	7
16	0.05	9.2352e-04	5.621e+02	20.961	9
17	0.05	1.9529e-03	5.691e+02	21.906	11
18	0.05	9.4417e-04	5.726e+02	22.304	8
19	0.04	3.6839e-04	5.739e+02	22.569	7
20	0.04	3.7166e-04	5.725e+02	22.342	6
21	0.04	5.8456e-04	5.704e+02	22.095	6
22	0.05	8.8278e-04	5.672e+02	21.742	7
23	0.05	4.1349e-04	5.687e+02	22.145	8
24	0.05	2.4727e-03	5.599e+02	21.493	9
25	0.04	6.1150e-04	5.577e+02	21.321	5
26	0.05	4.3317e-04	5.593e+02	21.625	7
27	0.05	3.9528e-04	5.607e+02	21.798	6
28	0.05	5.4531e-04	5.626e+02	21.823	4
29	0.05	2.5229e-04	5.635e+02	21.842	4
30	0.04	9.1549e-04	5.668e+02	21.834	4
31	0.05	1.9169e-03	5.600e+02	21.630	5
32	0.04	1.8207e-04	5.593e+02	21.565	4

	33		0.04		9.4024e-04		5.627e+02		21.601		4	
	34		0.04		6.0374e-04		5.605e+02		21.538		4	
	35		0.05		5.7683e-04		5.626e+02		21.898		8	
	36		0.05		1.0780e-03		5.587e+02		21.485		8	
	37		0.05		3.6849e-05		5.589e+02		21.460		4	

=====

Compute the perplexity of the held-out test set.

```
[~, ppl2] = logp(md12, 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.

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

```
rmpath(exampleFolder)
```

Input Arguments

ldaMd1 — 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 of the words of a single document.

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 appears in the *i*th document. Otherwise, the value `counts(i, j)` corresponds to the number of times the *i*th word 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.

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.

This option only applies if you specify the input documents as a matrix of word counts.

Data Types: `char`

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.

pp1 — Perplexity

positive scalar

Perplexity of the documents calculated from the log-probabilities, returned as a positive scalar.

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

Remove inflections from words 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"])

documents =
    2×1 tokenizedDocument:

(1,1) 6 tokens: a strongly worded collection of words
(2,1) 4 tokens: another collection of words
```

```
newDocuments = normalizeWords(documents)

newDocuments =
  2×1 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.

```
words = split("a strongly worded collection of words")
```

```
words = 6×1 string array
    "a"
    "strongly"
    "worded"
    "collection"
    "of"
    "words"
```

```
newWords = normalizeWords(words)
```

```
newWords = 6×1 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 `sonnetsDocuments` data. `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.

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

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

```
documents = sonnetsDocuments;
```

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
```

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

```
rmpath(exampleFolder)
```

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, 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, 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

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
    ...
```

Fit an LDA model with 20 topics.

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

Initial topic assignments sampled in 0.0365048 seconds.

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|         0 |    0.02 |      Inf | 1.159e+03 | 5.000 |         0 |
|         1 |    0.04 | 5.4884e-02 | 8.028e+02 | 5.000 |         0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |         0 |
|         3 |    0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |         0 |
|         4 |    0.03 | 3.4662e-03 | 7.430e+02 | 5.000 |         0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |         0 |
|         6 |    0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |         0 |
=====
```

```
mdl =
ldaModel with properties:
```

```
    NumTopics: 20
 WordConcentration: 1
 TopicConcentration: 5
```

```
CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
DocumentTopicProbabilities: [154x20 double]
TopicWordProbabilities: [3092x20 double]
Vocabulary: [1x3092 string]
FitInfo: [1x1 struct]
```

Predict the top topics for an array of new documents.

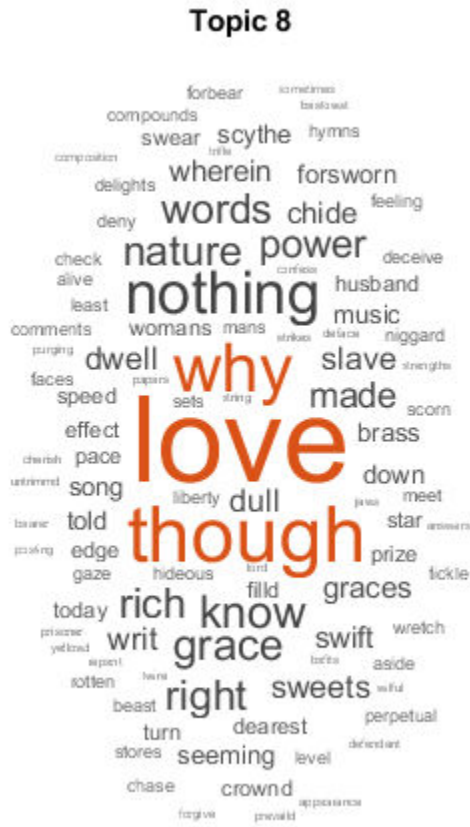
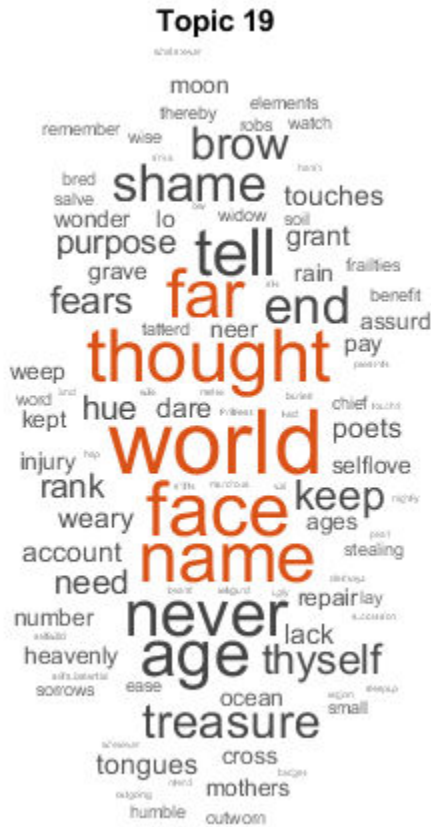
```
newDocuments = tokenizedDocument([
    "what's in a name? a rose by any other name would smell as sweet."
    "if music be the food of love, play on."]);
topicIdx = predict mdl, newDocuments

topicIdx =

    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))
```



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

```
rmpath(exampleFolder)
```

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 =
```

```
    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 0.0200075 seconds.

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
|=====|=====|=====|=====|=====|=====|
|         0 |    0.01 |      Inf | 1.159e+03 | 5.000 |         0 |
|         1 |    0.03 | 5.4884e-02 | 8.028e+02 | 5.000 |         0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |         0 |
|         3 |    0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |         0 |
|         4 |    0.02 | 3.4662e-03 | 7.430e+02 | 5.000 |         0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |         0 |
|         6 |    0.02 | 6.4180e-05 | 7.291e+02 | 5.000 |         0 |
|=====|=====|=====|=====|=====|=====|
```

```
mdl =
```

```
  ldaModel with properties:
```

```
          NumTopics: 20
          WordConcentration: 1
          TopicConcentration: 5
  CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
  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 =
```

```
3
15
19
3
14
```

Calculate Topic Prediction Scores

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
         ...
```

Fit an LDA model with 20 topics.

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

```
Initial topic assignments sampled in 0.0291698 seconds.
```

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
```

```
=====
|          0 |          0.01 |          Inf | 1.159e+03 | 5.000 | 0 |
|          1 |          0.03 | 5.4884e-02 | 8.028e+02 | 5.000 | 0 |
|          2 |          0.03 | 4.7400e-03 | 7.778e+02 | 5.000 | 0 |
|          3 |          0.03 | 3.4597e-03 | 7.602e+02 | 5.000 | 0 |
|          4 |          0.03 | 3.4662e-03 | 7.430e+02 | 5.000 | 0 |
|          5 |          0.03 | 2.9259e-03 | 7.288e+02 | 5.000 | 0 |
|          6 |          0.03 | 6.4180e-05 | 7.291e+02 | 5.000 | 0 |
=====
```

```
mdl =
  ldaModel with properties:

          NumTopics: 20
      WordConcentration: 1
      TopicConcentration: 5
  CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
  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 smell
iterationLimit = 200;
[topicIdx,scores] = predict(mdl,newDocument, ...
    'IterationLimit',iterationLimit)
```

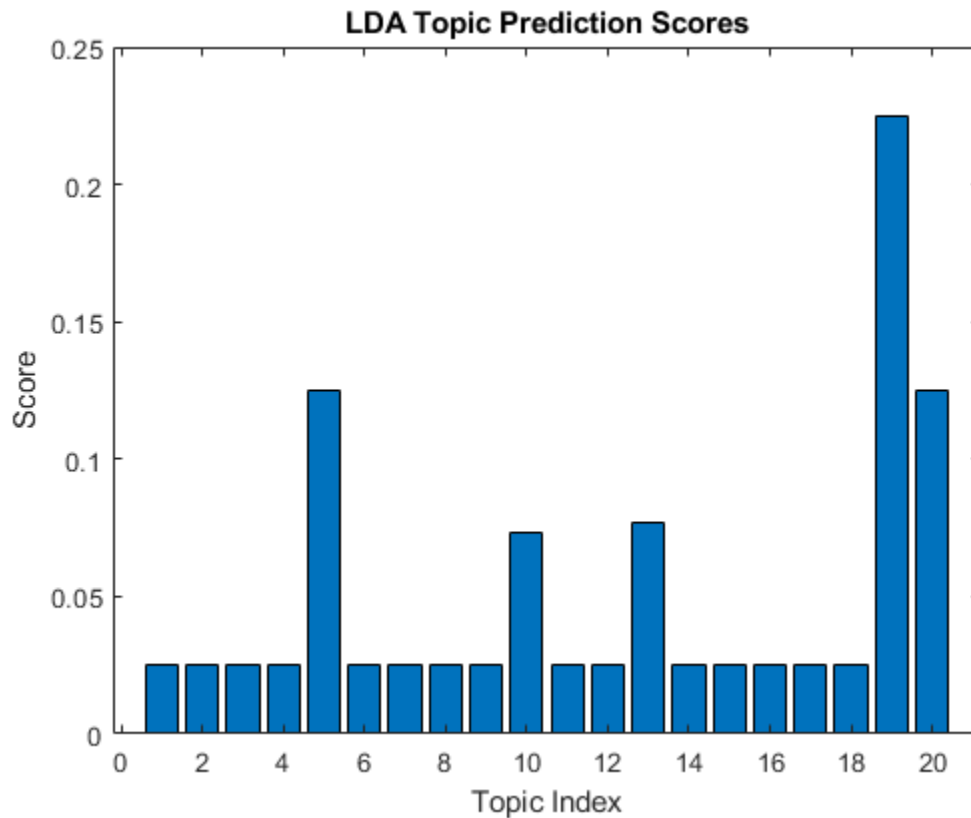
```
topicIdx = 19
```

```
scores =

    0.0250    0.0250    0.0250    0.0250    0.1250    0.0250    0.0250    0.0250    0.0250
```

View the prediction scores in a bar chart.

```
figure
bar(scores)
title("LDA Topic Prediction Scores")
xlabel("Topic Index")
ylabel("Score")
```



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

```
rmpath(exampleFolder)
```

Input Arguments

ldaMd1 — Input LDA model

ldaModel object

Input LDA model, specified as an `ldaModel` object.

documents — Input documents

tokenizedDocument array

Input documents, specified as a tokenizedDocument array.

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 appears in the *i*th document. Otherwise, the value `counts(i, j)` corresponds to the number of times the *i*th word 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.

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.

This option only applies if you specify the input documents as a matrix of word counts.

Data Types: char

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

extractFileText

Read text from PDF, Microsoft Word, and plain text files

Syntax

```
str = extractFileText(filename)
___ = extractFileText( ___, Name, Value)
```

Description

`str = extractFileText(filename)` reads the text data from a file as a string.

`___ = extractFileText(___, 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 ripper 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 Data Using a File Datastore

Import three example sonnet text files using a file datastore.

Create a file datastore for the example sonnet text files. The example sonnets have filenames "exampleSonnet1.txt", "exampleSonnet2.txt", and "exampleSonnet3.txt". Specify the read function to be `extractFileText`.

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

```
fds =
```

```
FileDatastore with properties:
```

```
Files: {
```

```
' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet1.t
```

```
' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet2.t
```

```
' ...\ib3A4F7C\26\tp410e6e2b\textanalytics-ex73762432\exampleSonnet3.t
```

```
}
```

```
UniformRead: 0
```

```
ReadFcn: @extractFileText
```

Create an empty bag-of-words model.

```
bag = bagOfWords
```

```
bag =
```

```
Empty bagOfWords
```

Loop over the files in the datastore and read each file. Tokenize the text 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 224 words and 3 documents:
```

	From	fairest	creatures	we	desire	...
	1	1	1	1	1	
	0	0	0	0	0	
	...					

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 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 use specify one explicitly.

This option only applies when the input is a plain text file.

Data Types: `char` | `string`

Password — Password to open PDF file

character vector | string scalar

Password to open PDF file, specified as 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 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

See Also

tokenizedDocument | writeTextDocument

Topics

“Extract Text Data From Files”

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

Introduced in R2017b

readWordEmbedding

Read word embedding from text file

Syntax

```
emb = readWordEmbedding(filename)
```

Description

`emb = readWordEmbedding(filename)` reads the pretrained word embedding stored in text file with name `filename`. The file encoding must be UTF-8 in either word2vec or GloVe text embedding 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

`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

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'.

Metacharacter	Description	Example
[c ₁ c ₂ c ₃]	Any character contained within the square brackets. The following characters are treated literally: \$. * + ? and - when not used to indicate a range.	'[rp.]ain' matches 'rain' or 'pain' or '.ain'.
[^c ₁ c ₂ c ₃]	Any character not contained within the square brackets. The following characters are treated literally: \$. * + ? and - when not used to indicate a range.	'[^*rp]ain' matches all four-letter sequences that end in 'ain', except 'rain' and 'pain' and '*ain'. For example, it matches 'gain', 'lain', or 'vain'.
[c ₁ -c ₂]	Any character in the range of c ₁ through c ₂	'[A-G]' matches a single character in the range of A through G.
\w	Any alphabetic, numeric, or underscore character. For English character sets, \w is equivalent to [a-zA-Z_0-9]	'\w*' identifies a word.
\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.
\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.	'' matches an HTML tag when the file name contains one or more characters.
expr{m,n}	At least m times, but no more than n times consecutively. {0,1} is equivalent to ?.	'\S{4,8}' matches between four and eight non-white-space characters.
expr{m,}	At least m times consecutively. {0,} and {1,} are equivalent to * and +, respectively.	'' 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
<code>exprq</code>	Greedy expression: match as many characters as possible.	Given the text ' <code><tr><td><p>text</p></td></code> ', the expression ' <code></?t.*></code> ' matches all characters between <code><tr</code> and <code>/td></code> : ' <code><tr><td><p>text</p></td></code> '
<code>exprq?</code>	Lazy expression: match as few characters as necessary.	Given the text ' <code><tr><td><p>text</p></td></code> ', the expression ' <code></?t.*?></code> ' ends each match at the first occurrence of the closing angle bracket (<code>></code>): ' <code><tr></code> ' ' <code><td></code> ' ' <code></td></code> '
<code>exprq+</code>	Possessive expression: match as much as possible, but do not rescan any portions of the text.	Given the text ' <code><tr><td><p>text</p></td></code> ', the expression ' <code></?t.*+></code> ' does not return any matches, because the closing angle bracket is captured using <code>.*</code> , 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
<code>(expr)</code>	Group elements of the expression and capture tokens.	' <code>Joh?n\s(\w*)</code> ' captures a token that contains the last name of any person with the first name John or Jon.

Grouping Operator	Description	Example
(?: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.
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
<code>expr (?=test)</code>	Look ahead for characters that match test.	<code>'\w*(?=ing)'</code> matches terms that are followed by <code>ing</code> , such as <code>'Fly'</code> and <code>'fall'</code> in the input text <code>'Flying, not falling.'</code>
<code>expr (?!test)</code>	Look ahead for characters that do not match test.	<code>'i(?!ng)'</code> matches instances of the letter <code>i</code> that are not followed by <code>ng</code> .
<code>(?<=test)expr</code>	Look behind for characters that match test.	<code>'(?<=re)\w*'</code> matches terms that follow <code>'re'</code> , such as <code>'new'</code> , <code>'use'</code> , and <code>'cycle'</code> in the input text <code>'renew, reuse, recycle'</code>
<code>(?<!test)expr</code>	Look behind for characters that do not match test.	<code>'(?<!\d)(\d)(?!\d)'</code> 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
<code>(?=test)expr</code>	Match both test and expr.	<code>'(?=[a-z])[^aeiou]'</code> matches consonants.
<code>(?!test)expr</code>	Match expr and do not match test.	<code>'(?![aeiou])[a-z]'</code> 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
<code>expr1 expr2</code>	Match expression <code>expr1</code> or expression <code>expr2</code> . If there is a match with <code>expr1</code> , then <code>expr2</code> is ignored.	' <code>(let tel)\w+</code> ' matches words that start with <code>let</code> or <code>tel</code> .
<code>(?(cond)expr)</code>	If condition <code>cond</code> is true, then match <code>expr</code> .	' <code>(?(?@ispc)[A-Z]:\\)</code> ' matches a drive name, such as <code>C:\</code> , when run on a Windows® system.
<code>(?(cond)expr1 expr2)</code>	If condition <code>cond</code> is true, then match <code>expr1</code> . Otherwise, match <code>expr2</code> .	' <code>Mr(s?)\..*?(?(1)her his)\w*</code> ' matches text that includes <code>her</code> when the text begins with <code>Mrs</code> , or that includes <code>his</code> when the text begins with <code>Mr</code> .

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
<code>(expr)</code>	Capture in a token the characters that match the enclosed expression.	' <code>Joh?n\s(\w*)</code> ' captures a token that contains the last name of any person with the first name <code>John</code> or <code>Jon</code> .
<code>\N</code>	Match the Nth token.	' <code><(\w+).*>.*</\1></code> ' captures tokens for HTML tags, such as <code>'title'</code> from the text <code>'<title>Some text</title>'</code> .
<code>(?(N)expr1 expr2)</code>	If the Nth token is found, then match <code>expr1</code> . Otherwise, match <code>expr2</code> .	' <code>Mr(s?)\..*?(?(1)her his)\w*</code> ' matches text that includes <code>her</code> when the text begins with <code>Mrs</code> , or that includes <code>his</code> when the text begins with <code>Mr</code> .

Named Token Operator	Description	Example
<code>(?<name>expr)</code>	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.
<code>\k<name></code>	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>'.
<code>(? (name) expr1 expr2)</code>	If the named token is found, then match <code>expr1</code> . Otherwise, match <code>expr2</code> .	'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 <i>expr</i> and include the resulting term in the match expression. When parsed, <i>expr</i> 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 <i>expr</i> , 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 <i>cmd</i> , and include the output returned by the command in the match expression.	'(.{2,}).?(??@flip1r(\$1)) ' finds palindromes that are at least four characters long, such as 'abba'.
(?@cmd)	Execute the MATLAB command represented by <i>cmd</i> , 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	Nth token
\$<name>	Named token
\${cmd}	Output returned when MATLAB executes the command, <i>cmd</i>

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
<code>(?-i)</code>	Match letter case (default for <code>regexp</code> and <code>regexprep</code>).
<code>(?i)</code>	Do not match letter case (default for <code>regexpi</code>).
<code>(?s)</code>	Match dot (<code>.</code>) in the pattern with any character (default).
<code>(?-s)</code>	Match dot in the pattern with any character that is not a newline character.
<code>(?-m)</code>	Match the <code>^</code> and <code>\$</code> metacharacters at the beginning and end of text (default).
<code>(?m)</code>	Match the <code>^</code> and <code>\$</code> metacharacters at the beginning and end of a line.
<code>(?-x)</code>	Include space characters and comments when matching (default).
<code>(?x)</code>	Ignore space characters and comments when matching. Use <code>'\ '</code> and <code>'\#'</code> to match space and <code>#</code> 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
<code>\$&</code> or <code>\$0</code>	Portion of the input text that is currently a match
<code>\$`</code>	Portion of the input text that precedes the current match
<code>\$'</code>	Portion of the input text that follows the current match (use <code>\$''</code> to represent <code>\$'</code>)
<code>\$N</code>	Nth token
<code>\$<name></code>	Named token
<code>\${cmd}</code>	Output returned when MATLAB executes the command, <code>cmd</code>

Operator	Description
<code>\a</code>	Alarm (beep)
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical tab
<code>\char</code>	Any character with special meaning in regular expressions that you want to match literally (for example, use <code>\\</code> to match a single backslash)

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

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

removeDocument

Remove documents from bag-of-words model

Syntax

```
newBag = removeDocument (bag, idx)
```

Description

`newBag = removeDocument (bag, idx)` removes the documents with indices specified by `idx` from the bag-of-words model `bag`. If the removed documents contain words that do not appear in the remaining documents, then the function also removes these words from the vocabulary.

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 9 words and 4 documents:
```

an	example	of	a	short	...
1	1	1	1	1	
0	0	0	1	1	
...					

Remove the first and third documents from `bag`.

```
idx = [1 3];  
newBag = removeDocument(bag, idx)
```

```
newBag =  
bagOfWords with 5 words and 2 documents:
```

	a	short	sentence	second	final
1		1	1	1	0
1		0	1	0	1

Remove the same documents using logical indices.

```
idx = logical([1 0 1 0]);  
newBag = removeDocument(bag, idx)
```

```
newBag =  
bagOfWords with 5 words and 2 documents:
```

	a	short	sentence	second	final
1		1	1	1	0
1		0	1	0	1

Input Arguments

bag — Input bag-of-words model

bagOfWords object

Input bag-of-words model, specified as a bagOfWords 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 bag-of-words model

bagOfWords object

Output bag-of-words model, returned as a bagOfWords object.

See Also

[addDocument](#) | [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 or bag-of-words 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 from the bag-of-words 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 7 words and 4 documents:

    an  example  of  a  short  ...
    1      1      1  1  1      1
    0      0      0  0      0
    ...
```

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

```
newBag = removeEmptyDocuments (bag)

newBag =
bagOfWords with 7 words and 2 documents:
```

an	example	of	a	short	...
1	1	1	1	1	1
0	0	0	1	1	

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
     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 model

bagOfWords object

Input bag-of-words model, specified as a bagOfWords object.

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.

idx — Indices of removed documents

vector of positive integers

Indices of removed documents, returned as a vector of positive integers.

See Also

`addDocument` | `bagOfWords` | `removeDocument` | `tokenizedDocument`

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

Introduced in R2017b

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 8 words and 4 documents:
```

an	example	of	a	short	...
1	1	1	1	1	
0	0	0	1	1	
...					

Remove the words that appear two times or fewer from the bag-of-words model.

```
count = 2;
newBag = removeInfrequentWords (bag, count)

newBag =
bagOfWords with 3 words and 4 documents:
```

example	a	short
1	1	1
0	1	1
...		

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.

Output Arguments

newBag — Output bag-of-words model

bagOfWords object

Output bag-of-words model, returned as a bagOfWords object.

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:
    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)
```

```
bag =
bagOfWords with 7 words and 2 documents:
```

an	example	of	a	short	...
1	1	1	1	1	1
0	0	0	1	1	

```
newBag = removeLongWords(bag, 7)
```

```
newBag =
bagOfWords with 5 words and 2 documents:
```

an	of	a	short	second
1	1	1	1	0
0	0	1	1	1

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

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:

  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)
```

```
bag =
bagOfWords with 7 words and 2 documents:
```

an	example	of	a	short	...
1	1	1	1	1	1
0	0	0	1	1	1

```
newBag = removeShortWords(bag, 2)
```

```
newBag =
bagOfWords with 4 words and 2 documents:
```

example	short	sentence	second
1	1	1	0
0	1	1	1

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 document 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"])

documents =
    2x1 tokenizedDocument:

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

Remove the stop words using the `stopWords` function to input a list of stop words directly into `removeWords`.

```
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)

bag =
bagOfWords with 7 words and 2 documents:
```

an	example	of	a	short	...
1	1	1	1	1	
0	0	0	1	1	

Remove the stop words using the `stopWords` function to input a list of stop words directly into `removeWords`.

```
newBag = removeWords(bag, stopWords)

newBag =
bagOfWords with 4 words and 2 documents:
```

example	short	sentence	second
1	1	1	0
0	1	1	1

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 4 words and 2 documents:
```

I	love	MATLAB	MathWorks
1	1	1	0
1	1	0	1

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 2 words and 2 documents:
```

love	MathWorks
1	0
1	1

Alternatively, you can specify logical indices.

```
idx = logical([1 0 1 0]);  
newBag = removeWords(bag,idx)  
  
newBag =  
bagOfWords with 2 words and 2 documents:
```

love	MathWorks
1	0
1	1

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 model `bag`. The input `bag` must be the same bag-of-words 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

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
         ...
```

Fit an LDA model with 20 topics. Set the LDA solver to be collapsed variational Bayes, zeroth order.

```
numTopics = 20;
solver = 'cvb0';
mdl = fitlda(bag,numTopics, ...
    'Solver',solver)
```

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
| 0 | 0.00 | Inf | 3.057e+03 | 5.000 | 0 |
| 1 | 0.02 | 1.4160e-01 | 1.130e+03 | 5.000 | 0 |
| 2 | 0.02 | 9.2528e-03 | 1.059e+03 | 5.000 | 0 |
| 3 | 0.02 | 1.3369e-02 | 9.662e+02 | 5.000 | 0 |
| 4 | 0.02 | 1.9319e-02 | 8.482e+02 | 5.000 | 0 |
| 5 | 0.02 | 2.4929e-02 | 7.199e+02 | 5.000 | 0 |
| 6 | 0.02 | 2.7516e-02 | 6.036e+02 | 5.000 | 0 |
| 7 | 0.02 | 2.5877e-02 | 5.136e+02 | 5.000 | 0 |
| 8 | 0.02 | 2.1398e-02 | 4.506e+02 | 5.000 | 0 |
| 9 | 0.02 | 1.6470e-02 | 4.082e+02 | 5.000 | 0 |
| 10 | 0.02 | 1.2497e-02 | 3.790e+02 | 5.000 | 0 |
| 11 | 0.02 | 9.5261e-03 | 3.583e+02 | 4.727 | 4 |
| 12 | 0.02 | 7.8403e-03 | 3.423e+02 | 4.378 | 4 |
| 13 | 0.02 | 6.2264e-03 | 3.302e+02 | 4.084 | 4 |
| 14 | 0.03 | 4.8817e-03 | 3.210e+02 | 3.845 | 4 |
| 15 | 0.02 | 3.7863e-03 | 3.141e+02 | 3.651 | 4 |
=====
```

	16		0.02		2.9332e-03		3.088e+02		3.492		4	
	17		0.02		2.3495e-03		3.047e+02		3.363		3	
	18		0.02		1.8551e-03		3.015e+02		3.255		3	
	19		0.02		1.4673e-03		2.990e+02		3.168		3	
	20		0.02		1.2225e-03		2.969e+02		3.096		3	
=====												
	Iteration		Time per		Relative		Training		Topic		Concentr.	
			iter., s		Delta log(L)		perplexity		concentr.		iterations	
=====												
	21		0.02		1.0323e-03		2.952e+02		3.035		3	
	22		0.02		8.5489e-04		2.937e+02		2.982		3	
	23		0.02		7.0038e-04		2.926e+02		2.935		3	
	24		0.02		5.5328e-04		2.917e+02		2.894		3	
	25		0.02		4.4525e-04		2.909e+02		2.857		3	
	26		0.02		3.7084e-04		2.903e+02		2.826		3	
	27		0.02		3.1369e-04		2.898e+02		2.800		3	
	28		0.02		2.5498e-04		2.894e+02		2.777		3	
	29		0.02		2.0217e-04		2.890e+02		2.758		3	
	30		0.02		1.6488e-04		2.888e+02		2.740		3	
	31		0.02		1.3737e-04		2.886e+02		2.724		3	
	32		0.02		1.2004e-04		2.884e+02		2.709		3	
	33		0.02		1.0827e-04		2.882e+02		2.697		3	
	34		0.02		9.9321e-05		2.880e+02		2.685		2	
=====												

```
mdl =
  ldaModel with properties:

        NumTopics: 20
      WordConcentration: 1
    TopicConcentration: 2.6853
 CorpusTopicProbabilities: [0.0797 0.0452 0.0450 0.0446 0.0531 0.0466 0.0558 0.054
 DocumentTopicProbabilities: [154x20 double]
   TopicWordProbabilities: [3092x20 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: 34
NegativeLogLikelihood: 5.4207e+04
Perplexity: 288.0198
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 | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|          34 |      0.00 |      Inf | 2.880e+02 | 2.685 | 0 |
|          35 |      0.02 | 9.0329e-05 | 2.879e+02 | 2.675 | 3 |
|          36 |      0.02 | 8.2414e-05 | 2.877e+02 | 2.665 | 2 |
|          37 |      0.02 | 7.3216e-05 | 2.876e+02 | 2.656 | 2 |
|          38 |      0.02 | 6.5418e-05 | 2.875e+02 | 2.648 | 2 |
|          39 |      0.02 | 5.8085e-05 | 2.874e+02 | 2.641 | 2 |
|          40 |      0.02 | 5.1839e-05 | 2.873e+02 | 2.634 | 2 |
|          41 |      0.02 | 4.6270e-05 | 2.873e+02 | 2.628 | 2 |
|          42 |      0.02 | 4.1188e-05 | 2.872e+02 | 2.622 | 2 |
|          43 |      0.02 | 3.6660e-05 | 2.871e+02 | 2.617 | 2 |
|          44 |      0.02 | 3.2955e-05 | 2.871e+02 | 2.612 | 2 |
|          45 |      0.02 | 3.0361e-05 | 2.870e+02 | 2.608 | 2 |
|          46 |      0.02 | 2.8804e-05 | 2.870e+02 | 2.604 | 2 |
|          47 |      0.02 | 2.7744e-05 | 2.869e+02 | 2.600 | 2 |
|          48 |      0.02 | 2.7177e-05 | 2.869e+02 | 2.596 | 2 |
|          49 |      0.02 | 2.7819e-05 | 2.868e+02 | 2.593 | 2 |
|          50 |      0.02 | 3.0082e-05 | 2.868e+02 | 2.589 | 2 |
|          51 |      0.02 | 3.4454e-05 | 2.867e+02 | 2.586 | 2 |
|          52 |      0.02 | 4.1612e-05 | 2.867e+02 | 2.582 | 2 |
|          53 |      0.02 | 5.5459e-05 | 2.866e+02 | 2.578 | 2 |
|          54 |      0.02 | 8.3476e-05 | 2.865e+02 | 2.574 | 2 |
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|          55 |      0.02 | 1.2059e-04 | 2.863e+02 | 2.570 | 2 |
|          56 |      0.02 | 1.2359e-04 | 2.861e+02 | 2.566 | 2 |
|          57 |      0.02 | 7.9046e-05 | 2.859e+02 | 2.562 | 2 |

```

	58		0.02		6.0826e-05		2.858e+02		2.558		2	
	59		0.02		7.7819e-05		2.857e+02		2.555		2	
	60		0.02		7.4205e-05		2.856e+02		2.551		2	
	61		0.02		5.8746e-05		2.855e+02		2.547		2	
	62		0.02		4.0434e-05		2.854e+02		2.542		2	
	63		0.02		3.2233e-05		2.854e+02		2.537		2	
	64		0.02		3.0283e-05		2.853e+02		2.533		2	
	65		0.02		2.9373e-05		2.853e+02		2.529		2	
	66		0.02		2.8480e-05		2.852e+02		2.525		2	
	67		0.02		2.7578e-05		2.852e+02		2.521		2	
	68		0.02		2.6618e-05		2.851e+02		2.518		2	
	69		0.02		2.5553e-05		2.851e+02		2.514		2	
	70		0.02		2.4375e-05		2.851e+02		2.511		2	
	71		0.02		2.3104e-05		2.850e+02		2.507		2	
	72		0.02		2.1771e-05		2.850e+02		2.504		2	
	73		0.02		2.0408e-05		2.850e+02		2.501		2	
	74		0.02		1.9043e-05		2.849e+02		2.498		2	

```
=====
```

Iteration		Time per		Relative		Training		Topic		Concentr.		
		iter., s		Delta log(L)		perplexity		concentr.		iterations		
	75		0.02		1.7698e-05		2.849e+02		2.495		2	
	76		0.02		1.6391e-05		2.849e+02		2.492		2	
	77		0.02		1.5135e-05		2.848e+02		2.490		2	
	78		0.02		1.3939e-05		2.848e+02		2.487		2	
	79		0.02		1.2810e-05		2.848e+02		2.485		2	
	80		0.02		1.1752e-05		2.848e+02		2.483		2	
	81		0.02		1.0767e-05		2.848e+02		2.481		2	
	82		0.02		9.8540e-06		2.848e+02		2.479		2	

```
=====
```

updatedMdl =

ldaModel with properties:

```

                NumTopics: 20
            WordConcentration: 1
        TopicConcentration: 2.4793
    CorpusTopicProbabilities: [0.1060 0.0441 0.0434 0.0435 0.0520 0.0444 0.0539 0.0533]
    DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 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: 82
    NegativeLogLikelihood: 5.4097e+04
    Perplexity: 284.7508
    Solver: "cvb0"
    History: [1x1 struct]
```

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

```
rmpath(exampleFolder)
```

Resume Fitting of LDA Model to 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 =
```

```
    154    3092
```

Fit an LDA model with 20 topics. Set the LDA solver to be approximate variational Bayes. For reproducibility, set `rng` to 'default'.

```
rng('default')
numTopics = 20;
solver = 'avb';
mdl = fitlda(counts,numTopics, ...
    'Solver','avb')
```

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
```

```
|         0 |         0.01 |         Inf | 3.057e+03 | 5.000 | 0 |
|         1 |         0.02 | 6.0352e-02 | 1.936e+03 | 5.000 | 0 |
|         2 |         0.01 | 1.4644e-01 | 7.363e+02 | 5.000 | 0 |
|         3 |         0.01 | 2.6551e-02 | 6.207e+02 | 5.000 | 0 |
|         4 |         0.01 | 1.6459e-02 | 5.593e+02 | 5.000 | 0 |
|         5 |         0.01 | 3.4489e-03 | 5.473e+02 | 5.000 | 0 |
|         6 |         0.01 | 2.5300e-03 | 5.386e+02 | 5.000 | 0 |
|         7 |         0.01 | 2.0317e-03 | 5.318e+02 | 5.000 | 0 |
|         8 |         0.01 | 7.2187e-04 | 5.294e+02 | 5.000 | 0 |
|         9 |         0.01 | 8.3417e-04 | 5.322e+02 | 5.000 | 0 |
|        10 |         0.01 | 1.3326e-03 | 5.278e+02 | 5.000 | 0 |
|        11 |         0.01 | 1.1235e-06 | 5.278e+02 | 7.845 | 11 |
=====
```

```
mdl =
  ldaModel with properties:

        NumTopics: 20
        WordConcentration: 1
        TopicConcentration: 7.8447
        CorpusTopicProbabilities: [0.0543 0.0543 0.0481 0.0525 0.0459 0.0573 0.0537 0.051
        DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
        Vocabulary: [1x3092 string]
        FitInfo: [1x1 struct]
```

Resume fitting the LDA model with a lower log-likelihood tolerance.

```
tolerance = 1e-5;
updatedMdl = resume(mdl, counts, ...
    'LogLikelihoodTolerance', tolerance)

updatedMdl =
  ldaModel with properties:

        NumTopics: 20
        WordConcentration: 1
        TopicConcentration: 11.5379
        CorpusTopicProbabilities: [0.0487 0.0540 0.0419 0.0457 0.0426 0.1056 0.0553 0.046
        DocumentTopicProbabilities: [154x20 double]
        TopicWordProbabilities: [3092x20 double]
        Vocabulary: [1x3092 string]
        FitInfo: [1x1 struct]
```


Input Arguments

ldaMdl — Input LDA model

ldaModel object

Input LDA model, specified as an ldaModel object.

bag — Input bag-of-words model

bagOfWords object

Input bag-of-words model, specified as a bagOfWords object.

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 appears in the *i*th document. Otherwise, the value counts(*i*,*j*) corresponds to the number of times the *i*th word 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.

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.

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.

This option only applies if you specify the input documents as a matrix of word counts.

Data Types: `char`

FitTopicConcentration — Option for fitting topic concentration parameter

`true` | `false`

Option for fitting topic concentration parameter, specified as the comma-separated pair consisting of `'FitTopicConcentration'` and either `true` or `false`. If `true`, then the function fits the scalar concentration parameter in the Dirichlet prior.

The default value is the value used to fit `lda`.

Example: `'FitTopicConcentration',true`

Data Types: `logical`

FitTopicProbabilities — Option for fitting topic probabilities

`true` | `false`

Option for fitting topic probabilities, specified as the comma-separated pair consisting of `'FitTopicProbabilities'` and either `true` or `false`. If `true`, then the function fits the topic probabilities.

The default value is the value used to fit `lda`.

Example: `'FitTopicProbabilities',true`

Data Types: `logical`

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

ValidationData — Validation data to monitor optimization convergence

[] (default) | bagOfWords 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, 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

updatedMd1 — Updated LDA model

ldaModel object (default)

Updated LDA model, specified as an ldaModel object.

See Also

`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

stopWords

Stop word list

Syntax

```
words = stopWords
```

Description

`words = stopWords` returns a string array of common words which can be removed from documents before analysis.

Examples

List of Stop Words

View list of stop words usually removed from documents before analysis. Reshape the output for readability.

```
words = stopWords;  
reshape(words, [], 5)
```

```
ans =
```

```
38×5 string array
```

```
"a"           "doesn't"    "in"          "she'll"     "we're"  
"about"       "doesnt"     "instead"     "shell"      "we've"  
"above"       "doing"      "into"        "should"     "weve"  
"across"      "done"       "is"          "since"      "were"  
"after"       "don't"      "isn't"       "so"         "what"  
"all"         "dont"       "isnt"        "some"       "what's"  
"along"       "during"     "it"          "such"       "whats"  
"also"        "each"       "it'll"       "than"       "when"
```

"am"	"either"	"itll"	"that"	"when's"
"an"	"for"	"it's"	"the"	"whens"
"and"	"from"	"its"	"their"	"where"
"any"	"given"	"let's"	"them"	"whether"
"are"	"had"	"lets"	"then"	"which"
"aren't"	"has"	"may"	"there"	"while"
"arent"	"have"	"me"	"therefore"	"who"
"as"	"having"	"more"	"these"	"who'll"
"at"	"he"	"most"	"they"	"wholl"
"be"	"he'd"	"much"	"this"	"who's"
"because"	"hed"	"must"	"those"	"whos"
"been"	"he'll"	"my"	"through"	"who've"
"before"	"her"	"no"	"to"	"whove"
"being"	"here"	"not"	"too"	"will"
"between"	"hers"	"now"	"towards"	"with"
"both"	"him"	"of"	"under"	"within"
"but"	"himself"	"on"	"until"	"without"
"by"	"his"	"one"	"us"	"won't"
"can"	"how"	"only"	"use"	"would"
"can't"	"how's"	"or"	"used"	"wouldn't"
"cant"	"hows"	"other"	"uses"	"you"
"cannot"	"however"	"our"	"using"	"you'd"
"could"	"i"	"out"	"very"	"youd"
"couldn't"	"i'd"	"over"	"want"	"you'll"
"couldnt"	"i'll"	"said"	"was"	"youll"
"did"	"i'm"	"says"	"wasn't"	"you're"
"didn't"	"im"	"see"	"wasnt"	"youre"
"didnt"	"i've"	"she"	"we"	"you've"
"do"	"ive"	"she'd"	"we'd"	"youve"
"does"	"if"	"shed"	"we'll"	"your"

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"])

documents =
  2x1 tokenizedDocument:
```

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

Remove the stop words using the `stopWords` function to input a list of stop words directly into `removeWords`.

```
newDocuments = removeWords(documents, stopWords)
```

```
newDocuments =
  2×1 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)
```

```
bag =
bagOfWords with 7 words and 2 documents:
```

an	example	of	a	short	...
1	1	1	1	1	1
0	0	0	1	1	

Remove the stop words using the `stopWords` function to input a list of stop words directly into `removeWords`.

```
newBag = removeWords(bag, stopWords)
```

```
newBag =
bagOfWords with 4 words and 2 documents:
```

example	short	sentence	second
---------	-------	----------	--------

1	1	1	0
0	1	1	1

See Also

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

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.

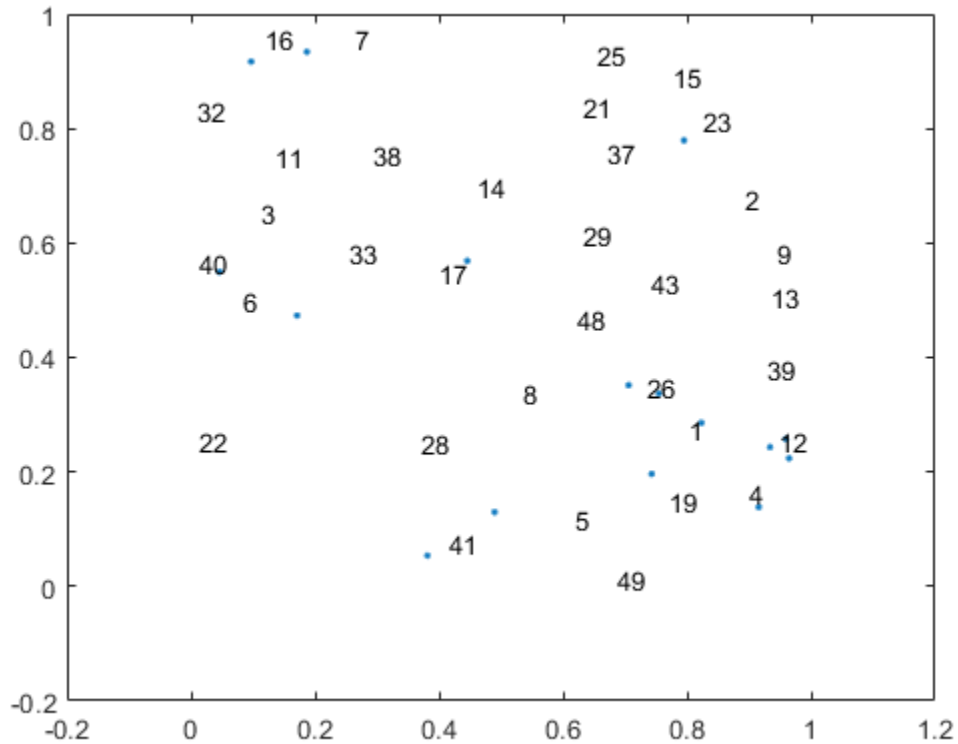
Examples

Create Text Scatter Plot

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

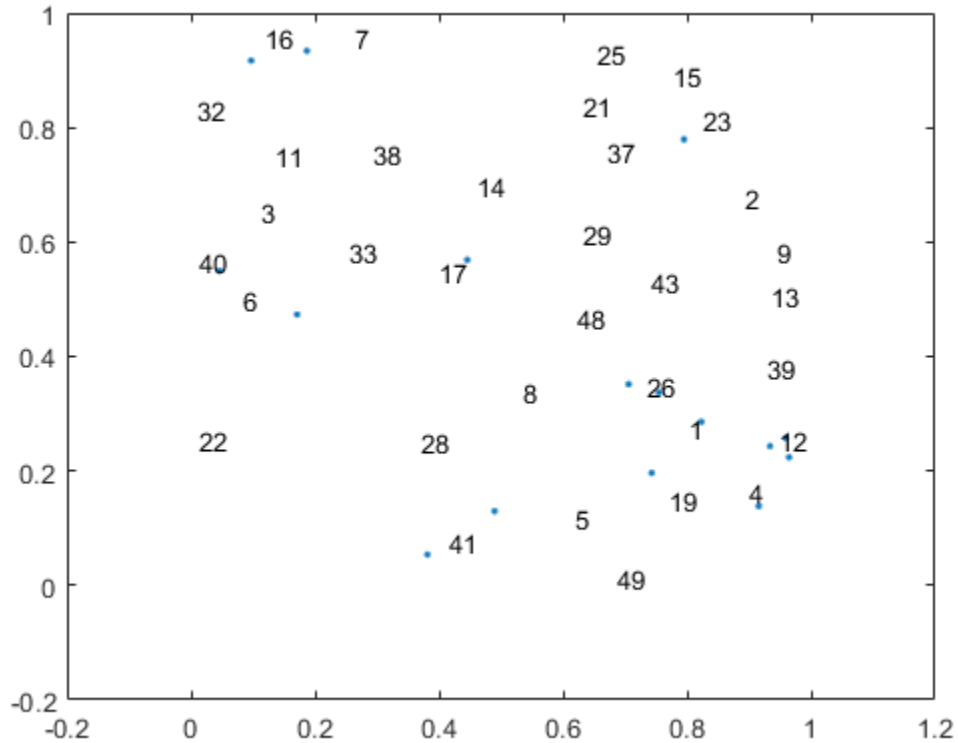
```
x = rand(50,1);
y = rand(50,1);
str = string(1:50);
```

```
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: [1×9999 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 =

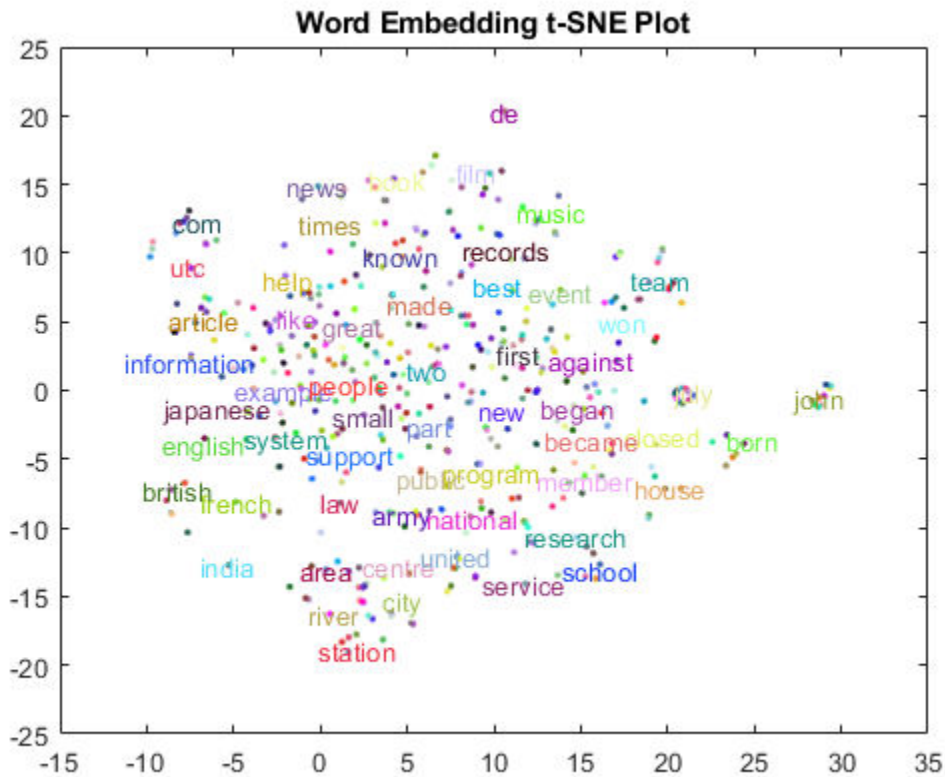
    500    50
```

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 by 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.

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.

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 `ax`.

`ts = textscatter3(___,Name,Value)` specifies additional `TextScatter` properties using one or more name-value pair arguments.

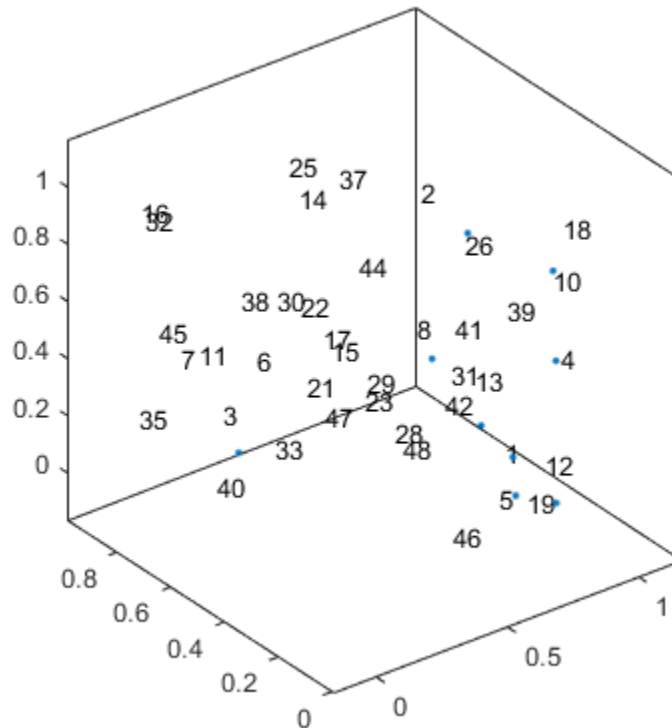
Examples

Create 3-D Text Scatter Plot

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

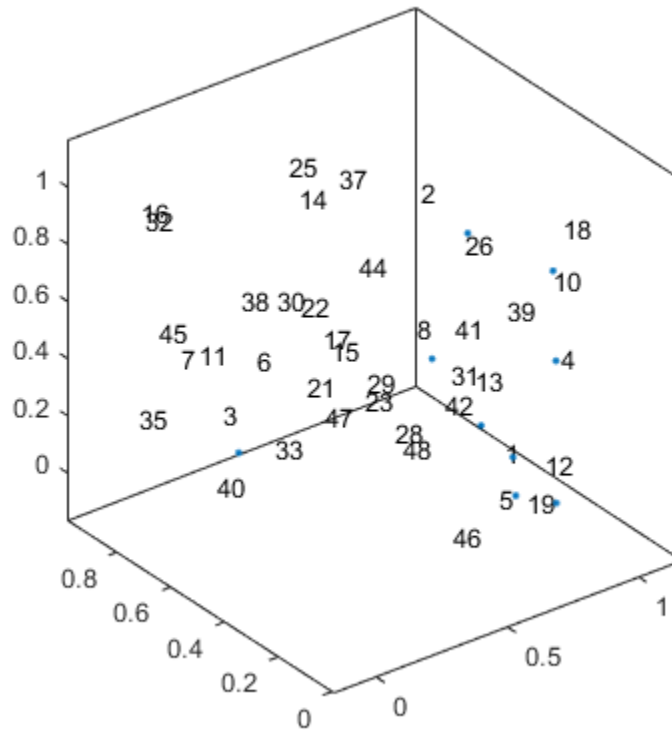
```
x = rand(50,1);
y = rand(50,1);
z = rand(50,1);
str = string(1:50);
```

```
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: [1×9999 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)
```

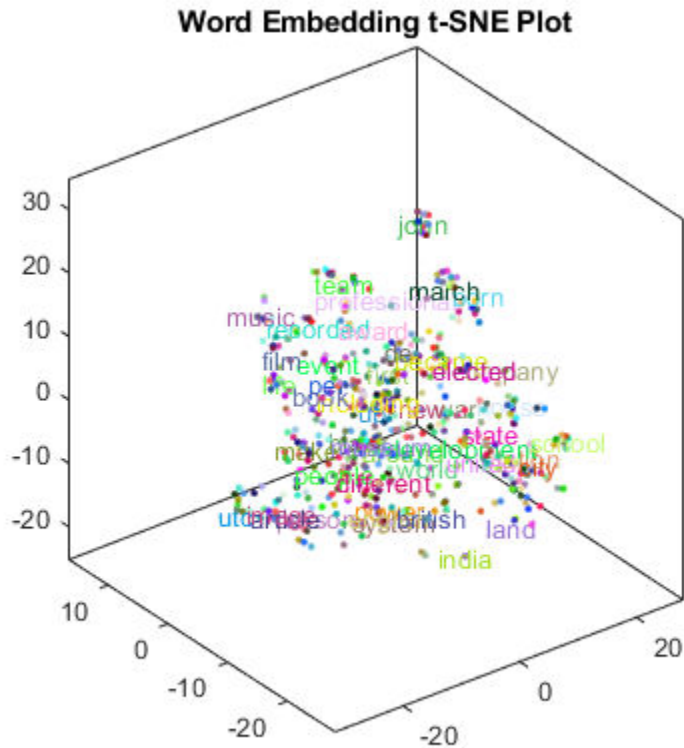
```
ans =
    500     50
```

Embed the word vectors in a three-dimensional 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 by 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.

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.

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

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 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 the bag-of-words model `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 `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.

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

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

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

```
bag =
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	1
0	0	0	0	0	0
...					

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

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

```
ans =
```

```
Columns 1 through 7
```

3.6507	4.3438	2.7344	3.6507	4.3438	2.2644	3.2452
0	0	0	0	0	4.5287	0
0	0	0	0	0	0	0
0	0	0	0	0	2.2644	0
0	0	0	0	0	2.2644	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	2.2644	0
0	0	2.7344	0	0	0	0

```
Columns 8 through 10
```

3.8918	2.4720	2.5520
0	0	0
0	0	2.5520
0	0	0
0	0	0
0	0	0
0	0	0

```

0         0         0
0         0     2.5520
0         0         0

```

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

```
rmpath(exampleFolder)
```

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

To use the example file `sonnetsDocuments.m`, add the example folder to the path.

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

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

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

```
bag =
bagOfWords with 3092 words and 154 documents:
```

```

    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
         ...

```

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
```

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

```
rmpath(exampleFolder)
```

Specify TF Weight Formulas

Specify the TF weight formula of a Term Frequency-Inverse Document Frequency (tf-idf) matrix.

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

To use the example file `sonnetsDocuments.m`, add the example folder to the path.

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

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

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

```
bag =
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	
0	0	0	0	0	
...					

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


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

```
ans =
```

```
Columns 1 through 7
```

```

3.6507    4.3438    2.7344    3.6507    4.3438    2.2644    3.2452
         0         0         0         0         0    4.5287         0
         0         0         0         0         0         0         0
         0         0         0         0         0         2.2644         0
         0         0         0         0         0         2.2644         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         2.2644         0
         0         0    2.7344         0         0         0         0

```

```
Columns 8 through 10
```

```

3.8918    2.4720    2.5520
         0         0         0
         0         0    2.5520
         0         0         0
         0         0         0
         0         0         0
         0         0         0
         0         0         0
         0         0    2.5520
         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 =
```

```
Columns 1 through 7
```

3.6507	4.3438	2.7344	3.6507	4.3438	2.2644	3.2452
0	0	0	0	0	2.2644	0
0	0	0	0	0	0	0
0	0	0	0	0	2.2644	0
0	0	0	0	0	2.2644	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	2.2644	0
0	0	2.7344	0	0	0	0

Columns 8 through 10

1.9459	2.4720	2.5520
0	0	0
0	0	2.5520
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	2.5520
0	0	0

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

```
rmpath(exampleFolder)
```

Input Arguments

bag — Input bag-of-words model

`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` 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 of the words of a single 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: `'Normalized', true` specifies to normalize the word 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 word counts.
- `'binary'` – Set the TF factor to the matrix of ones and zeros where the ones indicate whether a word is in a document.
- `'log'` – Set the TF factor to $1 + \log(\text{bag.Counts})$.

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 word which is equivalent to `sum(bag.Counts)`.

Example: `'IDFWeight', 'smooth'`

Data Types: `char`

Normalized — Option to normalize word counts

`false` (default) | `true`

Option to normalize word counts, specified as the comma-separated pair consisting of `'Normalized'` and `true` or `false`. If `true`, then the function normalizes each vector of word 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 word count matrix, specified as the comma-separated pair consisting of `'DocumentsIn'` and one of the following:

- `'rows'` – Return a matrix of word counts with rows corresponding to documents.
- `'columns'` – Return a transposed matrix of word counts with columns corresponding to documents.

Data Types: `char`

Output Arguments

M — Output Term Frequency-Inverse Document Frequency matrix

sparse matrix

Output Term Frequency-Inverse Document Frequency matrix, specified as a sparse matrix.

See Also

`bagOfWords` | `encode` | `topkwords`

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

Introduced in R2017b

topkwords

Most important words in bag-of-words model or LDA topic

Syntax

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

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

```
T = topkwords (ldaMdl, k, topicIdx)
```

```
T = topkwords (ldaMdl, k, topicIdx, 'Scaling', Scaling)
```

Description

`T = topkwords (bag)` returns a table of the five words with the largest word counts in bag-of-words model `bag`.

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

`T = 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`.

`T = topkwords (ldaMdl, k, topicIdx, 'Scaling', Scaling)` applies scaling to the LDA word scores.

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 `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.

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

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

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

```
bag =
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	
0	0	0	0	0	
...					

Find the top five words.

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

```
T=5x2 table
```

Word	Count
"thy"	281
"thou"	234
"love"	162
"thee"	161
"doth"	88

Find the top 20 words.

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

```
T=20x2 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

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

```
rmpath(exampleFolder)
```

Highest Probability Words of LDA Topic

Create a table of the words with highest probability of an 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);
```

Fit an LDA model with 20 topics.


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

Initial topic assignments sampled in 0.0249502 seconds.

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
|=====|=====|=====|=====|=====|=====|
|         0 |      0.00 |          Inf | 1.159e+03 | 5.000 |          0 |
|         1 |      0.03 | 5.4884e-02 | 8.028e+02 | 5.000 |          0 |
|         2 |      0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |          0 |
|         3 |      0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |          0 |
|         4 |      0.03 | 3.4662e-03 | 7.430e+02 | 5.000 |          0 |
|         5 |      0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |          0 |
|         6 |      0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |          0 |
|=====|=====|=====|=====|=====|=====|
```

```
mdl =
  ldaModel with properties:

      NumTopics: 20
      WordConcentration: 1
      TopicConcentration: 5
      CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
      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)
```

Find the top 20 words of the first topic and use inverse mean scaling on the scores.

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

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

```
rmpath(exampleFolder)
```

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.

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 $\text{Phi} \cdot (\log(\text{Phi}) - \text{mean}(\log(\text{Phi}), 1))$, where `Phi` corresponds to `ldaMdl.TopicWordProbabilities`.

Example: 'Scaling', 'inversemean'

Data Types: char

Output Arguments

T — Table of top words

table

Table of top words sorted in order of importance.

When the input is a bag-of-words model, the table has the following columns:

Word	Word specified as a string.
Count	The number of times the word appears in the bag-of-words model.

When the input is an LDA model, the table has the following columns:

Word	Word specified as a string.
Score	The word probability for the given LDA topic.

See Also

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

Topics

“Prepare Text Data for Analysis”

“Create Simple Text Model for Classification”

“Analyze Text Data Using Topic Models”

Introduced in R2017b

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 white-space.

`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.

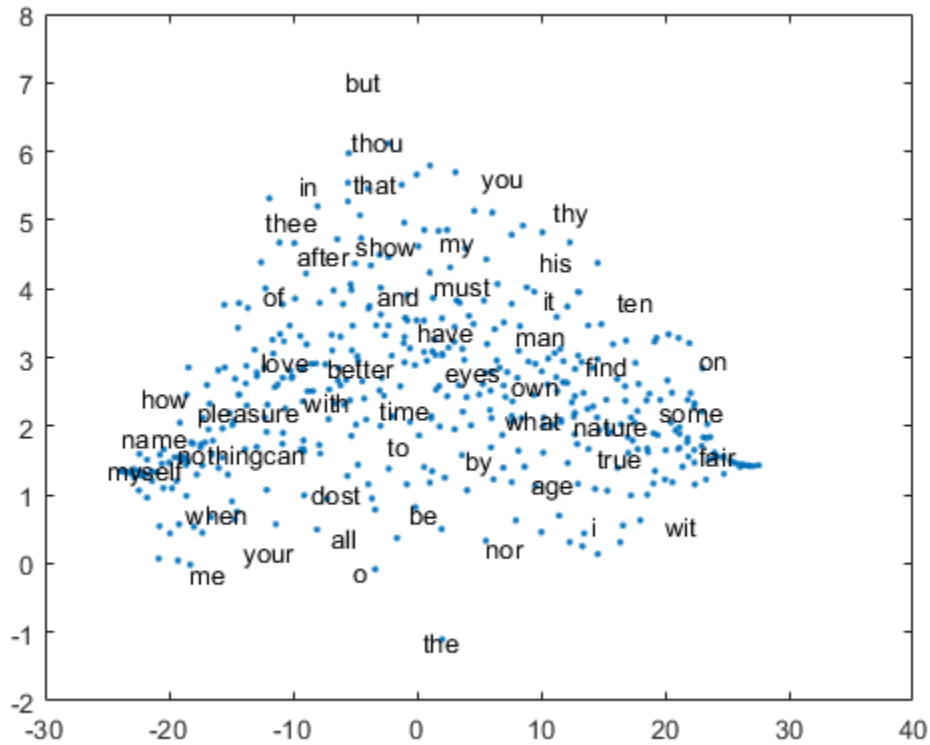
```
filename = "exampleSonnetsDocuments.txt";
emb = trainWordEmbedding(filename)
```

```
Training: 100% Loss: 2.65528 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 `sonnetsDocuments`. This function returns preprocessed versions of Shakespeare's sonnets as a `tokenizedDocument` array.

To use the example file `sonnetsDocuments.m`, add the example folder to the path.

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

Load the `sonnetsDocuments` data.

```
documents = sonnetsDocuments;  
emb = trainWordEmbedding(documents)
```

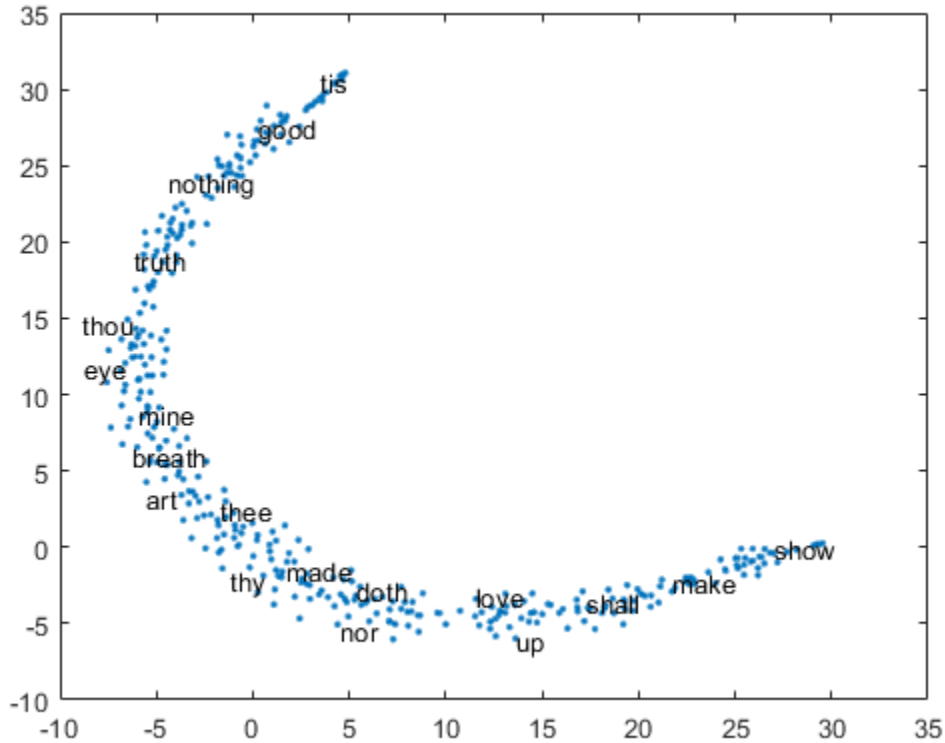
```
Training: 100% Loss: 0           Remaining time: 0 hours 0 minutes.
```

```
emb =  
wordEmbedding with properties:
```

```
    Dimension: 100  
    Vocabulary: [1x401 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)
```



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

```
rmpath(exampleFolder)
```

Specify Word Embedding Options

Train a word embedding using the example data `sonnetsDocuments`. This function returns preprocessed versions of Shakespeare's sonnets as a `tokenizedDocument` array.

To use the example file `sonnetsDocuments.m`, add the example folder to the path.

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

Load the `sonnetsDocuments` data.

```
documents = sonnetsDocuments;
```

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

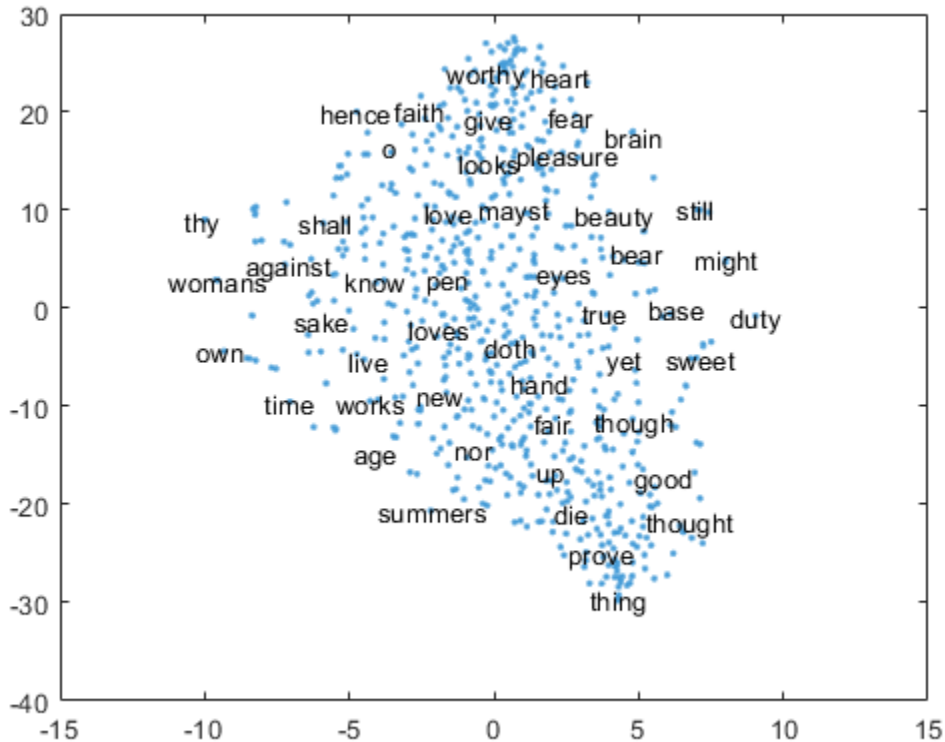
```
emb = trainWordEmbedding(documents, ...  
    'Dimension', 50, ...  
    'MinCount', 3, ...  
    'NumEpochs', 10)
```

```
Training: 100% Loss: 2.69927 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)
```

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

```
rmpath(exampleFolder)
```

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

`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, counts)
```

```
dscores = transform(ldaMdl, documents)
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, 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, 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 a `ldaModel` object.

Examples

Transform Documents into LSA Semantic Space

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.

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

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

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

```
bag =
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	
0	0	0	0	0	
...					

Fit an LSA model with 20 components.

```
numComponents = 20;
mdl = fitlsa(bag, numComponents)
```

```
mdl =
lsaModel with properties:
```

```
    NumComponents: 20
  ComponentWeights: [2.7866e+03 515.5889 443.6428 316.4191 295.4065 261.8927 2
  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 =
```

```
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
```

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

```
rmpath(exampleFolder)
```

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
```



```

0
0
0
0
0
...

```

Fit an LDA model with five topics.

```

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

```

Initial topic assignments sampled in 0.0249558 seconds.

```

=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|         0 |    0.01 |      Inf | 1.212e+03 | 1.250 |          0 |
|         1 |    0.02 | 1.2300e-02 | 1.112e+03 | 1.250 |          0 |
|         2 |    0.01 | 1.3254e-03 | 1.102e+03 | 1.250 |          0 |
|         3 |    0.01 | 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]

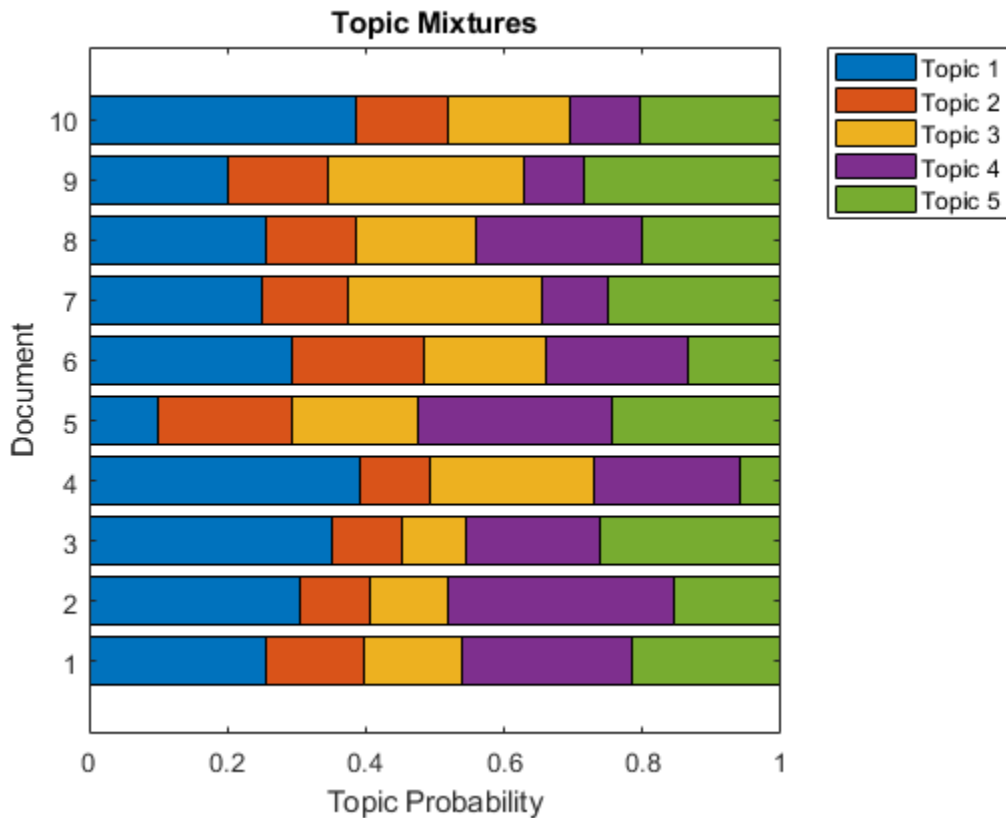
```

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 =
```

```
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 0.0210512 seconds.
```

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
|=====|=====|=====|=====|=====|=====|
|         0 |    0.01 |      Inf | 1.159e+03 | 5.000 |         0 |
|         1 |    0.03 | 5.4884e-02 | 8.028e+02 | 5.000 |         0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |         0 |
|         3 |    0.02 | 3.4597e-03 | 7.602e+02 | 5.000 |         0 |
|         4 |    0.03 | 3.4662e-03 | 7.430e+02 | 5.000 |         0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |         0 |
|         6 |    0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |         0 |
|=====|=====|=====|=====|=====|=====|
```

```
mdl =
```

```
ldaModel with properties:
```

```
    NumTopics: 20
    WordConcentration: 1
    TopicConcentration: 5
    CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
    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 =
```

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

lsaMd1 — Input LSA model

ldaModel object

Input LSA model, specified as an lsaModel object.

ldaMd1 — 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.

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 jth word appears in the ith document. Otherwise, the value counts(i, j) corresponds to the number of times the ith word appears in the jth 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 a `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.

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.

This option only applies if you specify the input documents as a matrix of word counts.

Data Types: char

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', 'euclidean')
```

Description

`words = vec2word(emb,M)` returns the closest words to the embedding vectors in the columns 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', 'euclidean')` uses the Euclidean 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`.

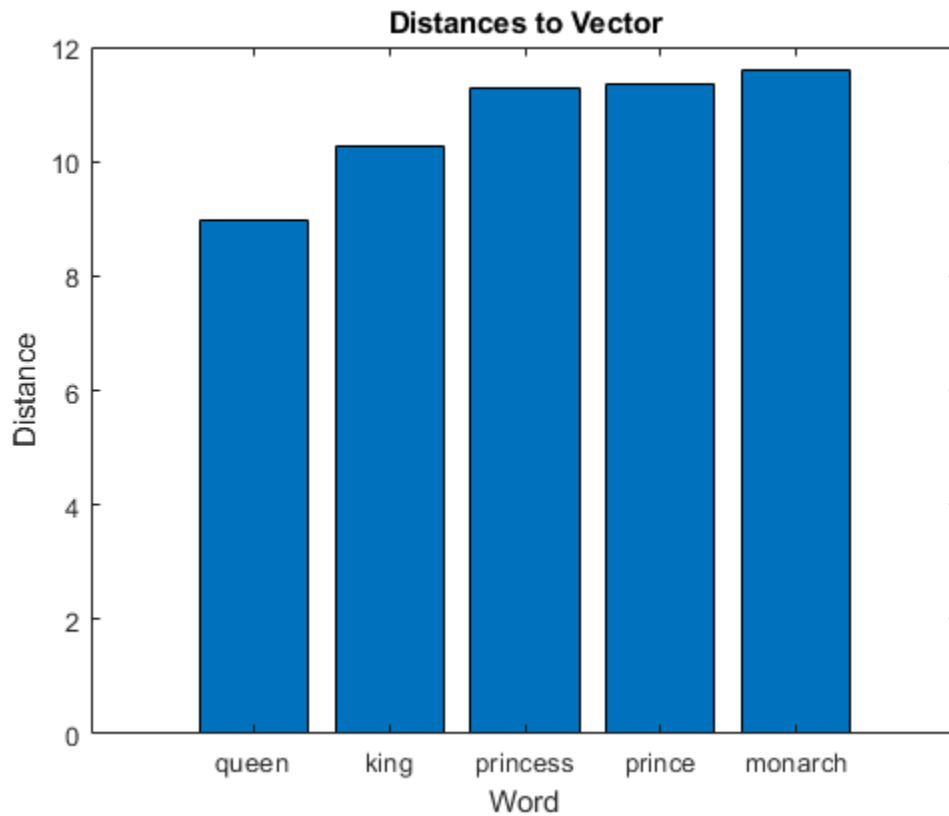
```
king = word2vec(emb, "king");
man = word2vec(emb, "man");
woman = word2vec(emb, "woman");
```

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 column of `M` corresponds to a word embedding vector. `M` must have `emb.Dimension` columns.

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: `'Distance', 'euclidean'` specifies to use the Euclidean distance metric.

Distance — Distance metric

`'cosine'` (default) | `'euclidean'`

Distance metric, specified as the comma-separated pair consisting of `'Distance'` and `'euclidean'` or `'cosine'`.

Example: `'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

`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 column of NaNs.

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 column of `M` corresponds to a word embedding vector. `M` has `emb.Dimension` columns.

See Also

`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 bag-of-words 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 and LDA topics. For the `wordcloud` (MATLAB) reference page, see `wordcloud`.

Syntax

```
wc = wordcloud(str)
wc = wordcloud(bag)
wc = wordcloud(ldaMdl,topicIdx)
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(bag)` creates a word cloud chart from the bag-of-words model `bag`.

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

`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");
```

View the first few characters of the text.

```
extractBefore(str,269)
```

```
ans =  
    "THE SONNETS  
  
    by William Shakespeare
```

```
I
```

```
From fairest creatures we desire increase,  
That thereby beauty's rose might never die,  
But as the ripper should by time decease,  
His tender heir might bear his memory:  
But thou, contracted to thine own bright eyes,"
```

Display the words from the sonnets in a word cloud.

```
figure  
wordcloud(str)
```

```
ans =  
    WordCloudChart with properties:  
  
        WordData: [1×2659 string]  
        SizeData: [1×2659 double]  
        MaxDisplayWords: 100
```

```
Show all properties
```



```
documents = sonnetsDocuments;
documents(1:10)

ans =
    10×1 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 issued
(10,1) 69 tokens: shame deny thou bearst love thy self art unprovident grant thou wilt
```

Create a bag-of-words model from the documents

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

```
bag =
bagOfWords with 3092 words and 154 documents:
```

fairest	creatures	desire	increase	thereby	...
1	1	1	1	1	
0	0	0	0	0	
...					

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

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



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

```
rmpath(exampleFolder)
```

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 3092 words and 154 documents:
```

```
    fairest    creatures    desire    increase    thereby    ...
         1         1         1         1         1
         0         0         0         0         0
    ...
```

Fit an LDA model with 20 topics.

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

Initial topic assignments sampled in 0.0208541 seconds.

```
=====
| Iteration | Time per | Relative | Training | Topic | Concentr. |
|           | iter., s | Delta log(L) | perplexity | concentr. | iterations |
=====
|         0 |    0.01 |      Inf | 1.159e+03 | 5.000 |          0 |
|         1 |    0.03 | 5.4884e-02 | 8.028e+02 | 5.000 |          0 |
|         2 |    0.03 | 4.7400e-03 | 7.778e+02 | 5.000 |          0 |
|         3 |    0.03 | 3.4597e-03 | 7.602e+02 | 5.000 |          0 |
|         4 |    0.03 | 3.4662e-03 | 7.430e+02 | 5.000 |          0 |
|         5 |    0.03 | 2.9259e-03 | 7.288e+02 | 5.000 |          0 |
|         6 |    0.03 | 6.4180e-05 | 7.291e+02 | 5.000 |          0 |
=====
```

```
mdl =
ldaModel with properties:
```

```
    NumTopics: 20
    WordConcentration: 1
    TopicConcentration: 5
    CorpusTopicProbabilities: [0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500 0.0500]
```

```
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`

bag — Input bag-of-words model

`bagOfWords` object

Input bag-of-words model, specified as a `bagOfWords` object.

ldaMd1 — 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: '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.

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]
'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.

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.

See Also

`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=8x2 table null
      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	Scalar string 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 <code>normalizeWords</code> . 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

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.

Train a word embedding using the example data returned by `sonnetsDocuments`. This function returns preprocessed versions of Shakespeare's sonnets as a `tokenizedDocument` array.

To use the example file `sonnetsDocuments.m`, add the example folder to the path.

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

Load the `sonnetsDocuments` data.

```
documents = sonnetsDocuments;  
emb = trainWordEmbedding(documents)
```

```
Training: 100% Loss: 2.78156 Remaining time: 0 hours 0 minutes.
```

```
emb =
  wordEmbedding with properties:

    Dimension: 100
    Vocabulary: [1×401 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: [1×401 string]
```

Remove the example folder from the path using rmpath.

```
rmpath(exampleFolder)
```

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

`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