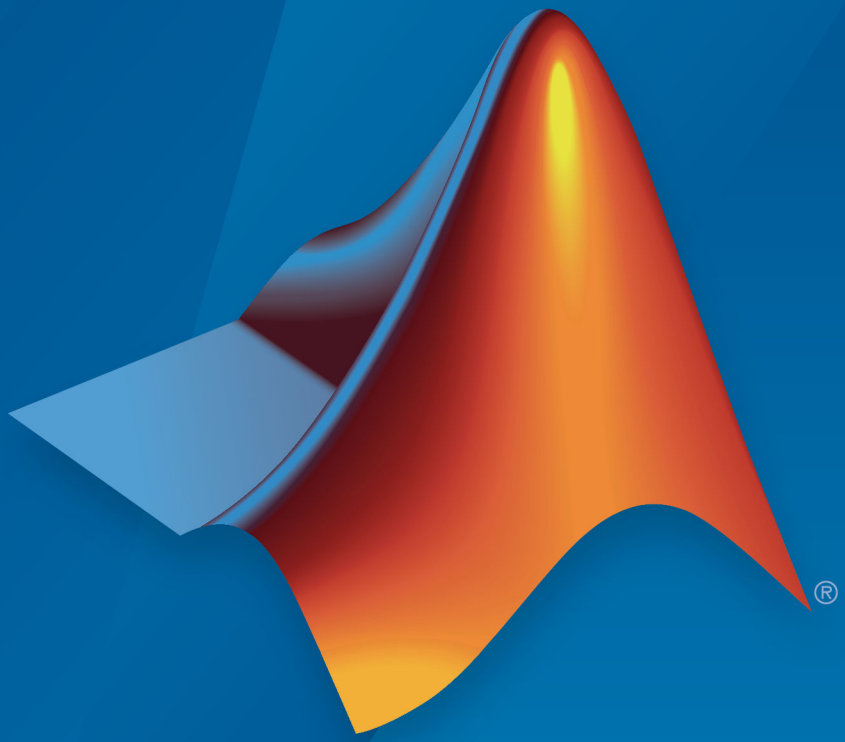
Text Analytics Toolbox™

User's Guide



MATLAB®

R2018b

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™ User's Guide

© COPYRIGHT 2017–2018 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

March 2018	Online Only	New for Version 1.1 (Release 2018a)
September 2018	Online Only	Revised for Version 1.2 (Release 2018b)

Contents

1 Text Data Preparation

Extract Text Data from Files **1-2**

Prepare Text Data for Analysis **1-11**

Parse HTML and Extract Text Content **1-21**

Language Support **1-27**
- Japanese Language Support **1-27**
- Language Considerations **1-27**
- Language-Independent Features **1-29**

Japanese Language Support **1-31**
- Tokenization **1-31**
- Stop Words **1-31**
- Part of Speech Details **1-32**
- Lemmatization **1-33**
- Language-Independent Features **1-34**

Analyze Japanese Text Data **1-35**

2 Modeling and Prediction

Create Simple Text Model for Classification **2-2**

Analyze Text Data Using Multiword Phrases **2-9**

Analyze Text Data Using Topic Models **2-18**

Analyze Text Data Containing Emojis **2-25**

Train a Sentiment Classifier **2-33**

Classify Text Data Using Deep Learning **2-43**

Pride and Prejudice and MATLAB **2-55**

Word-By-Word Text Generation Using Deep Learning **2-62**

Classify Out-of-Memory Text Data Using Custom Mini-Batch Datastore **2-69**

Choose Number of Topics for LDA Model **2-75**

Compare LDA Solvers **2-80**

3 Display and Presentation

Visualize Text Data Using Word Clouds **3-2**

Visualize Word Embeddings Using Text Scatter Plots **3-8**

Text Data Preparation

- “Extract Text Data from Files” on page 1-2
- “Prepare Text Data for Analysis” on page 1-11
- “Parse HTML and Extract Text Content” on page 1-21
- “Language Support” on page 1-27
- “Japanese Language Support” on page 1-31
- “Analyze Japanese Text Data” on page 1-35

Extract Text Data from Files

This example shows how to extract the text data from text, HTML, Microsoft® Word, PDF, CSV, and Microsoft Excel® files and import it into MATLAB® for analysis.

Usually, the easiest way to import text data into MATLAB is to use the `extractFileText` function. This function extracts the text data from text, PDF, HTML, and Microsoft Word files. To import text from CSV and Microsoft Excel files, use `readtable`. To extract text from HTML code, use `extractHTMLText`. To read data from PDF forms, use `readPDFFormData`.

Text File

Extract the text from `sonnets.txt` using `extractFileText`. The file `sonnets.txt` contains Shakespeare's sonnets in plain text.

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

View the first sonnet by extracting the text between the two titles "`I`" and "`II`".

```
start = " I" + newline;
fin = " II";
sonnet1 = extractBetween(str,start,fin)
```

```
sonnet1 =
    "
       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.

      "
```

Microsoft Word Document

Extract the text from `sonnets.docx` using `extractFileText`. The file `exampleSonnets.docx` contains Shakespeare's sonnets in a Microsoft Word document.

```
filename = "exampleSonnets.docx";
str = extractFileText(filename);
```

View the second sonnet by extracting the text between the two titles "`II`" and "`III`".

```
start = " II" + newline;
fin = " III";
sonnet2 = extractBetween(str,start,fin)
```

```
sonnet2 =
    "
       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.
```

```
    "
```

The example Microsoft Word document uses two newline characters between each line. To replace these characters with a single newline character, use the `replace` function.

```
sonnet2 = replace(sonnet2,[newline newline],newline)
```

```
sonnet2 =
    "
       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.
     "
```

PDF Files

Extract text from PDF documents and data from PDF forms.

Extract the text from `sonnets.pdf` using `extractFileText`. The file `exampleSonnets.pdf` contains Shakespeare's sonnets in a PDF.

```
filename = "exampleSonnets.pdf";
str = extractFileText(filename);
```

View the third sonnet by extracting the text between the two titles "`III`" and "`IV`". This PDF has a space before each newline character.

```
start = " III " + newline;
fin = "IV";
sonnet3 = extractBetween(str,start,fin)
```

```
sonnet3 =
    "
```

```
Look in thy glass and tell the face thou viewest
Now is the time that face should form another;
Whose fresh repair if now thou not renewest,
Thou dost beguile the world, unbless some mother.
For where is she so fair whose unear'd womb
Disdains the tillage of thy husbandry?
Or who is he so fond will be the tomb,
Of his self-love to stop posterity?
Thou art thy mother's glass and she in thee
Calls back the lovely April of her prime;
So thou through windows of thine age shalt see,
Despite of wrinkles this thy golden time.
  But if thou live, remember'd not to be,
  Die single and thine image dies with thee.


"
```

To read text data from PDF forms, use `readPDFFormData`. The function returns a struct containing the data from the PDF form fields.

```
filename = "weatherReportForm1.pdf";
data = readPDFFormData(filename)
```

```
data = struct with fields:
         event_type: "Thunderstorm Wind"
    event_narrative: "Large tree down between Plantersville and Nettleton."
```

HTML

Extract text from HTML files, HTML code, and the web.

To extract text data from a saved HTML file, use `extractFileText`.

```
filename = "exampleSonnets.html";
str = extractFileText(filename);
```

View the forth sonnet by extracting the text between the two titles `"IV"` and `"V"`.

```
start = newline + "IV" + newline;
fin = newline + "V" + newline;
sonnet4 = extractBetween(str,start,fin)
```

```
sonnet4 =
    "
     Unthrifty loveliness, why dost thou spend
      Upon thy self thy beauty's legacy?
      Nature's bequest gives nothing, but doth lend,
      And being frank she lends to those are free:
      Then, beauteous niggard, why dost thou abuse
      The bounteous largess given thee to give?
      Profitless usurer, why dost thou use
      So great a sum of sums, yet canst not live?
      For having traffic with thy self alone,
      Thou of thy self thy sweet self dost deceive:
      Then how when nature calls thee to be gone,
      What acceptable audit canst thou leave?
      Thy unused beauty must be tombed with thee,
      Which, used, lives th' executor to be.
     "
```

To extract text data from a string containing HTML code, use `extractHTMLText`.

```
code = "<html><body><h1>THE SONNETS</h1><p>by William Shakespeare</p></body></html>";
str = extractHTMLText(code)
```

```
str =
    "THE SONNETS

     by William Shakespeare"
```

To extract text data from a web page, first read the HTML code using `webread`, and then use `extractHTMLText`.

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

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

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

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

To find particular elements of HTML code, parse the code using `htmlTree` and use `findElement`. Parse the HTML code and find all the hyperlinks. The hyperlinks are nodes with element name `"A"`.

```
tree = htmlTree(code);
selector = "A";
subtrees = findElement(tree,selector);
```

View the first 10 subtrees and extract the text using `extractHTMLText`.

```
subtrees(1:10)
```

```
ans =
  10×1 htmlTree:

   (1,1)  <A class="svg_link navbar-brand" href="https://www.mathworks.com?s_ti…
   (2,1)  <A class="mwa-nav_login" href="https://www.mathworks.com/login?uri=ht…
   (3,1)  <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Product…
   (4,1)  <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solut…
   (5,1)  <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Acade…
   (6,1)  <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Suppor…
   (7,1)  <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Commu…
   (8,1)  <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">E…
   (9,1)  <A href="https://www.mathworks.com/company/aboutus/contact_us.html?s_…
  (10,1)  <A href="https://www.mathworks.com/store?s_cid=store_top_nav&amp;s_ti…
```

```
str = extractHTMLText(subtrees);
```

View the extracted text of the first 10 hyperlinks.

```
str(1:10)
```

```
ans = 10×1 string array
    ""
    "Sign In"
    "Products"
    "Solutions"
    "Academia"
    "Support"
    "Community"
    "Events"
    "Contact Us"
    "How to Buy"
```

To get the link targets, use `getAttributes` and specify the attribute `"href"` (hyperlink reference). Get the link targets of the first 10 subtrees.

```
attr = "href";
str = getAttribute(subtrees(1:10),attr)
```

```
str = 10×1 string array
    "https://www.mathworks.com?s_tid=gn_logo"
    "https://www.mathworks.com/login?uri=http://www.mathworks.com/help/textanalytics/ir
    "https://www.mathworks.com/products.html?s_tid=gn_ps"
    "https://www.mathworks.com/solutions.html?s_tid=gn_sol"
    "https://www.mathworks.com/academia.html?s_tid=gn_acad"
    "https://www.mathworks.com/support.html?s_tid=gn_supp"
    "https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc"
    "https://www.mathworks.com/company/events.html?s_tid=gn_ev"
    "https://www.mathworks.com/company/aboutus/contact_us.html?s_tid=gn_cntus"
    "https://www.mathworks.com/store?s_cid=store_top_nav&s_tid=gn_store"
```

CSV and Microsoft Excel Files

To extract text data from CSV and Microsoft Excel files, use `readtable` and extract the text data from the table that it returns.

Extract the text from the `events_narrative` column of `weatherReports.csv`.

```
T = readtable('weatherReports.csv','TextType','string');
head(T)
```

```
ans=8×16 table
            Time             event_id          state              event_type          da
    _____________________    _________    _______________    ____________________    __

    22-Jul-2016 16:10:00    6.4433e+05    "MISSISSIPPI"       "Thunderstorm Wind"
    15-Jul-2016 17:15:00    6.5182e+05    "SOUTH CAROLINA"    "Heavy Rain"
    15-Jul-2016 17:25:00    6.5183e+05    "SOUTH CAROLINA"    "Thunderstorm Wind"
    16-Jul-2016 12:46:00    6.5183e+05    "NORTH CAROLINA"    "Thunderstorm Wind"
    15-Jul-2016 14:28:00    6.4332e+05    "MISSOURI"          "Hail"
    15-Jul-2016 16:31:00    6.4332e+05    "ARKANSAS"          "Thunderstorm Wind"
    15-Jul-2016 16:03:00    6.4343e+05    "TENNESSEE"         "Thunderstorm Wind"
    15-Jul-2016 17:27:00    6.4344e+05    "TENNESSEE"         "Hail"
```

```
str = T.event_narrative;
str(1:10)
```

```
ans = 10×1 string array
    "Large tree down between Plantersville and Nettleton."
    "One to two feet of deep standing water developed on a street on the Winthrop Unive
    "NWS Columbia relayed a report of trees blown down along Tom Hall St."
    "Media reported two trees blown down along I-40 in the Old Fort area."
    ""
    "A few tree limbs greater than 6 inches down on HWY 18 in Roseland."
    "Awning blown off a building on Lamar Avenue. Multiple trees down near the intersec
    "Quarter size hail near Rosemark."
    "Tin roof ripped off house on Old Memphis Road near Billings Drive. Several large t
    "Powerlines down at Walnut Grove and Cherry Lane roads."
```

Extract Text from Multiple Files

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

Create a file datastore for the example sonnet text files. The examples files are named "`exampleSonnetN.txt`", where `N` is the number of the sonnet. Specify the file name using the wildcard "*" to find all file names of this structure. To specify the read function to be `extractFileText`, input this function to `fileDatastore` using a function handle.

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

```
fds =
  FileDatastore with properties:

                      Files: {
                             ' ...\Documents\MATLAB\examples\textanalytics-ex15735454\
                             ' ...\Documents\MATLAB\examples\textanalytics-ex15735454\
                             ' ...\Documents\MATLAB\examples\textanalytics-ex15735454\
                              ... and 1 more
                             }
                UniformRead: 0
                    ReadFcn: @extractFileText
    AlternateFileSystemRoots: {}
```

Loop over the files in the datastore and read each text file.

```
str = [];
while hasdata(fds)
    textData = read(fds);
```

```
    str = [str; textData];
end
```

View the extracted text.

```
str
```

```
str = 4×1 string array
    "  From fairest creatures we desire increase,↵  That thereby beauty's rose might ne
    "  When forty winters shall besiege thy brow,↵  And dig deep trenches in thy beauty
    "  Look in thy glass and tell the face thou viewest↵  Now is the time that face sho
    "  Unthrifty loveliness, why dost thou spend↵  Upon thy self thy beauty's legacy?↵
```

See Also

`extractFileText` | `extractHTMLText` | `readPDFFormData` | `tokenizedDocument`

Related Examples

- “Prepare Text Data for Analysis” on page 1-11
- “Create Simple Text Model for Classification” on page 2-2
- “Visualize Text Data Using Word Clouds” on page 3-2
- “Analyze Text Data Containing Emojis” on page 2-25
- “Analyze Text Data Using Topic Models” on page 2-18
- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Classify Text Data Using Deep Learning” on page 2-43
- “Train a Sentiment Classifier” on page 2-33

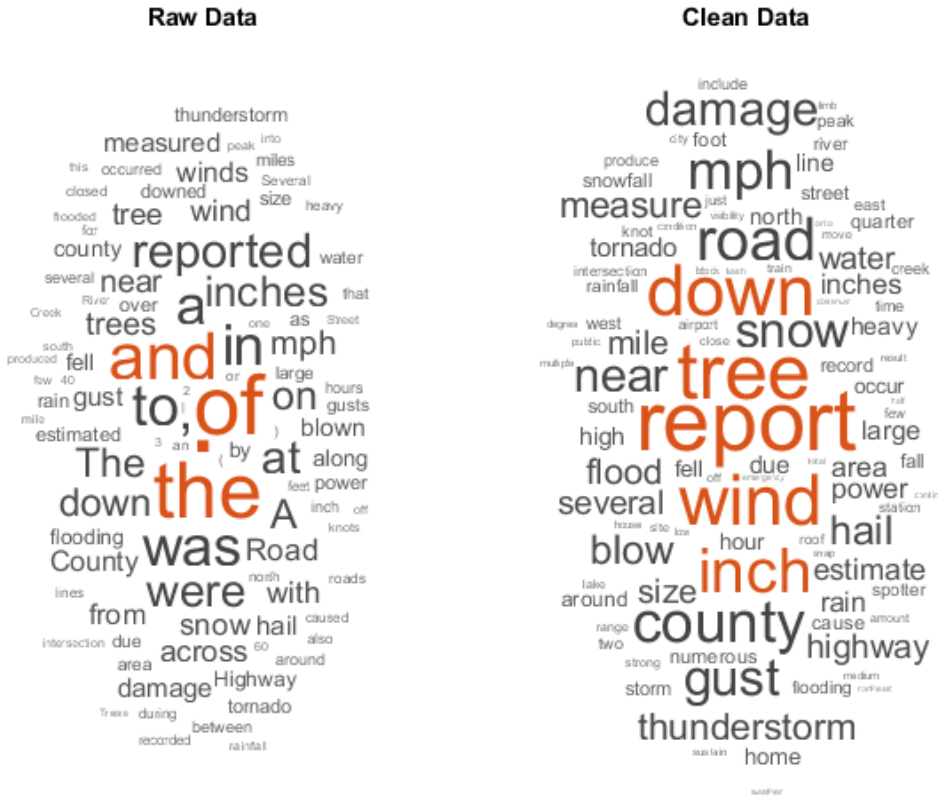
Prepare Text Data for Analysis

This example shows how to create a function which cleans and preprocesses text data for analysis.

Text data can be large and can contain lots of noise which negatively affects statistical analysis. For example, text data can contain the following:

- Variations in case, for example "new" and "New"
- Variations in word forms, for example "walk" and "walking"
- Words which add noise, for example stop words such as "the" and "of"
- Punctuation and special characters
- HTML and XML tags

These word clouds illustrate word frequency analysis applied to some raw text data from weather reports, and a preprocessed version of the same text data.



Load and Extract Text Data

Load the example data. The file `weatherReports.csv` contains weather reports, including a text description and categorical labels for each event.

```
filename = "weatherReports.csv";
data = readtable(filename,'TextType','string');
```

Extract the text data from the field `event_narrative`, and the label data from the field `event_type`.

```
textData = data.event_narrative;
labels = data.event_type;
textData(1:10)
```

```
ans = 10×1 string array
    "Large tree down between Plantersville and Nettleton."
    "One to two feet of deep standing water developed on a street on the Winthrop Unive
    "NWS Columbia relayed a report of trees blown down along Tom Hall St."
    "Media reported two trees blown down along I-40 in the Old Fort area."
    ""
    "A few tree limbs greater than 6 inches down on HWY 18 in Roseland."
    "Awning blown off a building on Lamar Avenue. Multiple trees down near the intersec
    "Quarter size hail near Rosemark."
    "Tin roof ripped off house on Old Memphis Road near Billings Drive. Several large t
    "Powerlines down at Walnut Grove and Cherry Lane roads."
```

Create Tokenized Documents

Convert the text data to lowercase.

```
cleanTextData = lower(textData);
cleanTextData(1:10)
```

```
ans = 10×1 string array
    "large tree down between plantersville and nettleton."
    "one to two feet of deep standing water developed on a street on the winthrop unive
    "nws columbia relayed a report of trees blown down along tom hall st."
    "media reported two trees blown down along i-40 in the old fort area."
    ""
    "a few tree limbs greater than 6 inches down on hwy 18 in roseland."
    "awning blown off a building on lamar avenue. multiple trees down near the intersec
    "quarter size hail near rosemark."
    "tin roof ripped off house on old memphis road near billings drive. several large t
    "powerlines down at walnut grove and cherry lane roads."
```

Create an array of tokenized documents.

```
cleanDocuments = tokenizedDocument(cleanTextData);
cleanDocuments(1:10)
```

```
ans =
  10×1 tokenizedDocument:

   (1,1)   8 tokens: large tree down between plantersville and nettleton .
   (2,1)  39 tokens: one to two feet of deep standing water developed on a stre…
   (3,1)  14 tokens: nws columbia relayed a report of trees blown down along to…
   (4,1)  14 tokens: media reported two trees blown down along i-40 in the old …
```

```
   (5,1)   0 tokens:
   (6,1)  15 tokens: a few tree limbs greater than 6 inches down on hwy 18 in r…
   (7,1)  20 tokens: awning blown off a building on lamar avenue . multiple tre…
   (8,1)   6 tokens: quarter size hail near rosemark .
   (9,1)  21 tokens: tin roof ripped off house on old memphis road near billing…
  (10,1)  10 tokens: powerlines down at walnut grove and cherry lane roads .
```

Erase the punctuation from the documents.

```
cleanDocuments = erasePunctuation(cleanDocuments);
cleanDocuments(1:10)
```

```
ans =
  10×1 tokenizedDocument:

   (1,1)   7 tokens: large tree down between plantersville and nettleton
   (2,1)  37 tokens: one to two feet of deep standing water developed on a stre…
   (3,1)  13 tokens: nws columbia relayed a report of trees blown down along to…
   (4,1)  13 tokens: media reported two trees blown down along i40 in the old f…
   (5,1)   0 tokens:
   (6,1)  14 tokens: a few tree limbs greater than 6 inches down on hwy 18 in r…
   (7,1)  18 tokens: awning blown off a building on lamar avenue multiple trees…
   (8,1)   5 tokens: quarter size hail near rosemark
   (9,1)  19 tokens: tin roof ripped off house on old memphis road near billing…
  (10,1)   9 tokens: powerlines down at walnut grove and cherry lane roads
```

Words like "a", "and", "to", and "the" (known as stop words) can add noise to data. Remove a list of stop words using the `removeStopWords` function.

```
cleanDocuments = removeStopWords(cleanDocuments);
cleanDocuments(1:10)
```

```
ans =
  10×1 tokenizedDocument:

   (1,1)   5 tokens: large tree down plantersville nettleton
   (2,1)  18 tokens: two feet deep standing water developed street winthrop uni…
   (3,1)  10 tokens: nws columbia relayed report trees blown down tom hall st
   (4,1)  10 tokens: media reported two trees blown down i40 old fort area
   (5,1)   0 tokens:
   (6,1)  10 tokens: few tree limbs greater 6 inches down hwy 18 roseland
   (7,1)  13 tokens: awning blown off building lamar avenue multiple trees down…
   (8,1)   5 tokens: quarter size hail near rosemark
```

```
   (9,1)  16 tokens: tin roof ripped off house old memphis road near billings d…
  (10,1)   7 tokens: powerlines down walnut grove cherry lane roads
```

Remove words with 2 or fewer characters, and words with 15 or greater characters.

```
cleanDocuments = removeShortWords(cleanDocuments,2);
cleanDocuments = removeLongWords(cleanDocuments,15);
cleanDocuments(1:10)
```

```
ans =
  10×1 tokenizedDocument:

   (1,1)   5 tokens: large tree down plantersville nettleton
   (2,1)  18 tokens: two feet deep standing water developed street winthrop uni…
   (3,1)   9 tokens: nws columbia relayed report trees blown down tom hall
   (4,1)  10 tokens: media reported two trees blown down i40 old fort area
   (5,1)   0 tokens:
   (6,1)   8 tokens: few tree limbs greater inches down hwy roseland
   (7,1)  13 tokens: awning blown off building lamar avenue multiple trees down…
   (8,1)   5 tokens: quarter size hail near rosemark
   (9,1)  16 tokens: tin roof ripped off house old memphis road near billings d…
  (10,1)   7 tokens: powerlines down walnut grove cherry lane roads
```

Lemmatize the words using `normalizeWords`. To improve lemmatization, first add part of speech details to the documents using `addPartOfSpeechDetails`.

```
cleanDocuments = addPartOfSpeechDetails(cleanDocuments);
cleanDocuments = normalizeWords(cleanDocuments,'Style','lemma');
cleanDocuments(1:10)
```

```
ans =
  10×1 tokenizedDocument:

   (1,1)   5 tokens: large tree down plantersville nettleton
   (2,1)  18 tokens: two foot deep standing water develop street winthrop unive…
   (3,1)   9 tokens: nws columbia relayed report tree blow down tom hall
   (4,1)  10 tokens: medium report two tree blow down i40 old fort area
   (5,1)   0 tokens:
   (6,1)   8 tokens: few tree limb great inches down hwy roseland
   (7,1)  13 tokens: awning blow off building lamar avenue multiple tree down n…
   (8,1)   5 tokens: quarter size hail near rosemark
   (9,1)  16 tokens: tin roof rip off house old memphis road near billings driv…
  (10,1)   7 tokens: powerlines down walnut grove cherry lane road
```

Create Bag-of-Words Model

Create a bag-of-words model.

```
cleanBag = bagOfWords(cleanDocuments)
```

```
cleanBag =
  bagOfWords with properties:

          Counts: [36176×18410 double]
      Vocabulary: [1×18410 string]
        NumWords: 18410
    NumDocuments: 36176
```

Remove words that do not appear more than two times in the bag-of-words model.

```
cleanBag = removeInfrequentWords(cleanBag,2)
```

```
cleanBag =
  bagOfWords with properties:

          Counts: [36176×6952 double]
      Vocabulary: [1×6952 string]
        NumWords: 6952
    NumDocuments: 36176
```

Some preprocessing steps such as `removeInfrequentWords` leaves empty documents in the bag-of-words model. To ensure that no empty documents remain in the bag-of-words model after preprocessing, use `removeEmptyDocuments` as the last step.

Remove empty documents from the bag-of-words model and the corresponding labels from `labels`.

```
[cleanBag,idx] = removeEmptyDocuments(cleanBag);
labels(idx) = [];
cleanBag
```

```
cleanBag =
  bagOfWords with properties:

          Counts: [28137×6952 double]
      Vocabulary: [1×6952 string]
        NumWords: 6952
```

```
    NumDocuments: 28137
```

Create a Preprocessing Function

It can be useful to create a function which performs preprocessing so you can prepare different collections of text data in the same way. For example, you can use a function so that you can preprocess new data using the same steps as the training data.

Create a function which tokenizes and preprocesses the text data so it can be used for analysis. The function `preprocessWeatherNarratives`, performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Lemmatize the words using `normalizeWords`.

Use the example preprocessing function `preprocessWeatherNarratives` to prepare the text data.

```
newText = "A tree is downed outside Apple Hill Drive, Natick";
newDocuments = preprocessWeatherNarratives(newText)
```

```
newDocuments =
  tokenizedDocument:

   7 tokens: tree down outside apple hill drive natick
```

Compare with Raw Data

Compare the preprocessed data with the raw data.

```
rawDocuments = tokenizedDocument(textData);
rawBag = bagOfWords(rawDocuments)
```

```
rawBag =
  bagOfWords with properties:
```

```
          Counts: [36176×23302 double]
      Vocabulary: [1×23302 string]
        NumWords: 23302
    NumDocuments: 36176
```

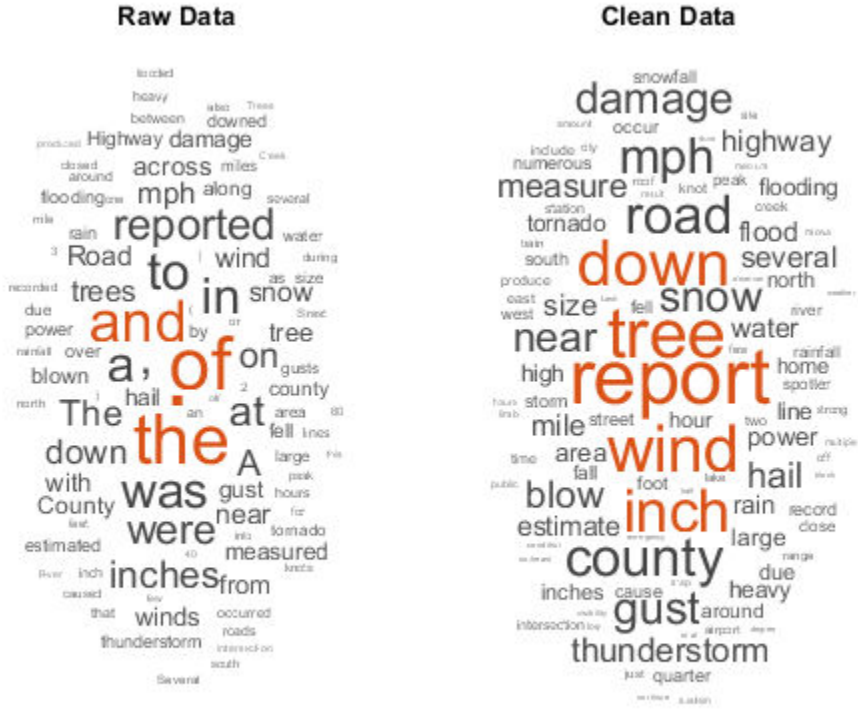
Calculate the reduction in data.

```
numWordsClean = cleanBag.NumWords;
numWordsRaw = rawBag.NumWords;
reduction = 1 - numWordsClean/numWordsRaw
```

```
reduction = 0.7017
```

Compare the raw data and the cleaned data by visualizing the two bag-of-words models using word clouds.

```
figure
subplot(1,2,1)
wordcloud(rawBag);
title("Raw Data")
subplot(1,2,2)
wordcloud(cleanBag);
title("Clean Data")
```



Example Preprocessing Function

The function `preprocessWeatherNarratives`, performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Lemmatize the words using `normalizeWords`.

```
function [documents] = preprocessWeatherNarratives(textData)
% Convert the text data to lowercase.
cleanTextData = lower(textData);

% Tokenize the text.
documents = tokenizedDocument(cleanTextData);

% Erase punctuation.
documents = erasePunctuation(documents);

% Remove a list of stop words.
documents = removeStopWords(documents);

% Remove words with 2 or fewer characters, and words with 15 or greater
% characters.
documents = removeShortWords(documents,2);
documents = removeLongWords(documents,15);

% Lemmatize the words.
documents = addPartOfSpeechDetails(documents);
documents = normalizeWords(documents,'Style','lemma');
end
```

See Also

`addPartOfSpeechDetails` | `bagOfWords` | `erasePunctuation` | `normalizeWords` | `removeEmptyDocuments` | `removeInfrequentWords` | `removeLongWords` | `removeShortWords` | `removeStopWords` | `tokenizedDocument` | `wordcloud`

Related Examples

- “Extract Text Data from Files” on page 1-2
- “Create Simple Text Model for Classification” on page 2-2
- “Visualize Text Data Using Word Clouds” on page 3-2
- “Analyze Text Data Containing Emojis” on page 2-25
- “Analyze Text Data Using Topic Models” on page 2-18
- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Classify Text Data Using Deep Learning” on page 2-43
- “Train a Sentiment Classifier” on page 2-33

Parse HTML and Extract Text Content

This example shows how to parse HTML code and extract the text content from particular elements.

Parse HTML Code

Read HTML code from the URL `https://www.mathworks.com/help/textanalytics` using `webread`.

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

Parse the HTML code using `htmlTree`.

```
tree = htmlTree(code);
```

View the HTML element name of the tree.

```
tree.Name
```

```
ans =
"HTML"
```

View the child elements of the tree. The children are subtrees of `tree`.

```
tree.Children
```

```
ans =
  4×1 htmlTree:

    " "
    <HEAD><TITLE>Text Analytics Toolbox Documentation</TITLE><META charset="utf-8"/><ME
    " "
    <BODY id="responsive_offcanvas"><!-- Mobile TopNav: Start --><DIV class="header vis
```

Extract Text from HTML Tree

To extract text directly from the HTML tree, use `extractHTMLText`.

```
str = extractHTMLText(tree)
```

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

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

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

Find HTML Elements

To find particular elements of an HTML tree, use `findElement`. Find all the hyperlinks in the HTML tree. In HTML, hyperlinks use the "A" tag.

```
selector = "A";
subtrees = findElement(tree,selector);
```

View the first few subtrees.

```
subtrees(1:20)
```

```
ans =
  20×1 htmlTree:

    <A class="svg_link navbar-brand" href="https://www.mathworks.com?s_tid=gn_logo"><IM
    <A class="mwa-nav_login" href="https://www.mathworks.com/login?uri=http://www.mathw
    <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Products</A>
    <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solutions</A>
    <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Academia</A>
    <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Support</A>
    <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Community</A>
    <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">Events</A>
    <A href="https://www.mathworks.com/company/aboutus/contact_us.html?s_tid=gn_cntus">
    <A href="https://www.mathworks.com/store?s_cid=store_top_nav&amp;s_tid=gn_store">Ho
    <A href="https://www.mathworks.com/company/aboutus/contact_us.html?s_tid=gn_cntus">
    <A href="https://www.mathworks.com/store?s_cid=store_top_nav&amp;s_tid=gn_store">Ho
    <A class="mwa-nav_login" href="https://www.mathworks.com/login?uri=http://www.mathw
    <A class="svg_link pull-left" href="https://www.mathworks.com?s_tid=gn_logo"><IMG a
    <A href="https://www.mathworks.com/products.html?s_tid=gn_ps">Products</A>
    <A href="https://www.mathworks.com/solutions.html?s_tid=gn_sol">Solutions</A>
    <A href="https://www.mathworks.com/academia.html?s_tid=gn_acad">Academia</A>
    <A href="https://www.mathworks.com/support.html?s_tid=gn_supp">Support</A>
    <A href="https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc">Community</A>
    <A href="https://www.mathworks.com/company/events.html?s_tid=gn_ev">Events</A>
```

Create a word cloud from the text of the hyperlinks.

```
str = extractHTMLText(subtrees);
figure
wordcloud(str);
title("Hyperlinks")
```

Get HTML Attributes

Get the class attributes from the paragraph elements in the HTML tree.

```
subtrees = findElement(tree,'p');
attr = "class";
str = getAttribute(subtrees,attr)
```

```
str = 21×1 string array
    <missing>
```

```
    <missing>
    "add_margin_5"
    <missing>
    <missing>
    <missing>
    <missing>
    <missing>
    "category_desc"
    "category_desc"
    "category_desc"
    "category_desc"
    <missing>
    <missing>
    <missing>
    "text-center"
    <missing>
    <missing>
    <missing>
    "copyright"
    <missing>
```

Create a word cloud from the text contained in paragraph elements with class `"category_desc"`.

```
subtrees = findElement(tree,'p.category_desc');
str = extractHTMLText(subtrees);
figure
wordcloud(str);
```

See Also

`extractHTMLText` | `findElement` | `getAttribute` | `htmlTree` | `tokenizedDocument`

Related Examples

- “Prepare Text Data for Analysis” on page 1-11
- “Create Simple Text Model for Classification” on page 2-2
- “Visualize Text Data Using Word Clouds” on page 3-2
- “Analyze Text Data Using Topic Models” on page 2-18

- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Classify Text Data Using Deep Learning” on page 2-43
- “Train a Sentiment Classifier” on page 2-33

Language Support

Text Analytics Toolbox supports the languages English and Japanese. Most Text Analytics Toolbox functions also work with text in other languages.

Japanese Language Support

For more information about analyzing Japanese text using Text Analytics Toolbox, see “Japanese Language Support” on page 1-31.

Language Considerations

This table summarizes how to use Text Analytics Toolbox features for other languages.

Feature	Language Consideration	Workaround
Tokenization	The `tokenizedDocument` function has built-in rules for English and Japanese only. For English text, the `'unicode'` tokenization method of `tokenizedDocument` detects tokens using rules based on Unicode® Standard Annex #29 [1] and the ICU tokenizer [2], modified to better detect complex tokens such as hashtags and URLs. For Japanese text, the `'mecab'` tokenization method detects tokens using rules based on the MeCab tokenizer [3].	For other languages, you can still try using `tokenizedDocument`. If `tokenizedDocument` does not produce useful results, then try tokenizing the text manually. To create a `tokenizedDocument` array from manually tokenized text, set the `'TokenizeMethod'` option to `'none'`. For more information, see `tokenizedDocument`.
Stop word removal	The `removeStopWords` function removes English and Japanese stop words only.	To remove stop words from other languages, use `removeWords` and specify your own stop words to remove.

Feature	Language Consideration	Workaround
Sentence detection	The `addSentenceDetails` function detects sentence boundaries based on punctuation characters and line number information. For English text, the function also uses a list of abbreviations passed to the function.	For other languages, you might need to specify your own list of abbreviations for sentence detection. To do this, use the `'Abbreviations'` option of `addSentenceDetails`. For more information, see `addSentenceDetails`.
Word clouds	For string input, the `wordcloud` function uses English and Japanese tokenization, stop word removal, and word normalization.	For other languages, you might need to manually preprocess your text data and specify unique words and corresponding sizes in `wordcloud`. To specify word sizes in `wordcloud`, input your data as a table or arrays containing the unique words and corresponding sizes. For more information, see `wordcloud`.

Feature	Language Consideration	Workaround
Word embeddings	File input to the `trainWordEmbedding` function requires words separated by whitespace.	For files containing non-English text, you might need to input a `tokenizedDocument` array to `trainWordEmbedding`. To create a `tokenizedDocument` array from pretokenized text, use the `tokenizedDocument` function and set the `'TokenizeMethod'` option to `'none'`. For more information, see `trainWordEmbedding`.

Language-Independent Features

Word and N-Gram Counting

The `bagOfWords` and `bagOfNgrams` functions support `tokenizedDocument` input regardless of language. If you have a `tokenizedDocument` array containing your data, then you can use these functions.

Modeling and Prediction

The `fitlda` and `fitlsa` functions support `bagOfWords` and `bagOfNgrams` input regardless of language. If you have a `bagOfWords` or `bagOfNgrams` object containing your data, then you can use these functions.

The `trainWordEmbedding` function supports `tokenizedDocument` or file input regardless of language. If you have a `tokenizedDocument` array or a file containing your data in the correct format, then you can use this function.

References

[1] *Unicode Text Segmentation*. https://www.unicode.org/reports/tr29/

[2] *Boundary Analysis*. http://userguide.icu-project.org/boundaryanalysis

[3] *MeCab: Yet Another Part-of-Speech and Morphological Analyzer*. https://taku910.github.io/mecab/

See Also

`addLanguageDetails` | `addSentenceDetails` | `bagOfNgrams` | `bagOfWords` | `fitlda` | `fitlsa` | `normalizeWords` | `removeWords` | `stopWords` | `tokenizedDocument` | `wordcloud`

More About

- “Text Data Preparation”
- “Modeling and Prediction”
- “Display and Presentation”
- “Japanese Language Support” on page 1-31
- “Analyze Japanese Text Data” on page 1-35

Japanese Language Support

This topic summarizes the Text Analytics Toolbox features that support Japanese text. For an example showing how to analyze Japanese text data, see “Analyze Japanese Text Data” on page 1-35.

Tokenization

The `tokenizedDocument` function automatically detects Japanese input. Alternatively, set the `'Language'` option in `tokenizedDocument` to `'ja'`. This option specifies the language details of the tokens. To view the language details of the tokens, use `tokenDetails`. These language details determine the behavior of the `removeStopWords`, `addPartOfSpeechDetails`, `normalizeWords`, and `addSentenceDetails` functions on the tokens.

Tokenize Japanese Text

Tokenize Japanese text using `tokenizedDocument`. The function automatically detects Japanese text.

```
str = [
    "恋に悩み、苦しむ。"
    "恋の悩みで苦しむ。"
    "空に星が輝き、瞬いている。"
    "空の星が輝きを増している。"];
documents = tokenizedDocument(str)
```

```
documents =
  4x1 tokenizedDocument:

     6 tokens: 恋 に 悩み 、 苦しむ 。
     6 tokens: 恋 の 悩み で 苦しむ 。
    10 tokens: 空 に 星 が 輝き 、 瞬い て いる 。
    10 tokens: 空 の 星 が 輝き を 増し て いる 。
```

Stop Words

To remove stop words from documents according to the token language details, use `removeStopWords`. For a list of Japanese stop words set the `'Language'` option in `stopWords` to `'ja'`.

Remove Japanese Stop Words

Tokenize Japanese text using `tokenizedDocument`. The function automatically detects Japanese text.

```
str = [
    "ここは静かなので、とても穏やかです"
    "企業内の顧客データを利用し、今年の売り上げを調べることが出来た。"
    "私は先生です。私は英語を教えています。"];
documents = tokenizedDocument(str);
```

Remove stop words using `removeStopWords`. The function uses the language details from `documents` to determine which language stop words to remove.

```
documents = removeStopWords(documents)
```

```
documents =
  3x1 tokenizedDocument:

     4 tokens: 静か 、 とても 穏やか
    10 tokens: 企業 顧客 データ 利用 、 今年 売り上げ 調べる 出来 。
     5 tokens: 先生 。 英語 教え 。
```

Part of Speech Details

The `tokenDetails` function, by default, includes part of speech details with the token details.

Get Part of Speech Details of Japanese Text

Tokenize Japanese text using `tokenizedDocument`.

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

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

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

```
ans=8×7 table
     Token      DocumentNumber    LineNumber        Type         Language    PartOfSpeech
    ________    ______________    __________    ____________    ________    ____________

    "恋"              1               1          letters           ja         noun
    "に"              1               1          letters           ja         adposition
    "悩み"            1               1          letters           ja         verb
    "、"              1               1          punctuation       ja         punctuation
    "苦しむ"          1               1          letters           ja         verb
    "。"              1               1          punctuation       ja         punctuation
    "恋"              2               1          letters           ja         noun
    "の"              2               1          letters           ja         adposition
```

Lemmatization

To lemmatize tokens according to the token language details, use `normalizeWords` and set the `'Style'` option to `'lemma'`.

Lemmatize Japanese Text

Tokenize Japanese text using the `tokenizedDocument` function. The function automatically detects Japanese text.

```
str = [
    "空に星が輝き、瞬いている。"
    "空の星が輝きを増している。"
    "駅までは遠くて、歩けない。"
    "遠くの駅まで歩けない。"];
documents = tokenizedDocument(str);
```

Lemmatize the tokens using `normalizeWords`.

```
documents = normalizeWords(documents)
```

```
documents =
  4x1 tokenizedDocument:
```

```
10 tokens: 空 に 星 が 輝く 、 瞬く て いる 。
10 tokens: 空 の 星 が 輝き を 増す て いる 。
 9 tokens: 駅 まで は 遠い て 、 歩ける ない 。
 7 tokens: 遠く の 駅 まで 歩ける ない 。
```

Language-Independent Features

Word and N-Gram Counting

The `bagOfWords` and `bagOfNgrams` functions support `tokenizedDocument` input regardless of language. If you have a `tokenizedDocument` array containing your data, then you can use these functions.

Modeling and Prediction

The `fitlda` and `fitlsa` functions support `bagOfWords` and `bagOfNgrams` input regardless of language. If you have a `bagOfWords` or `bagOfNgrams` object containing your data, then you can use these functions.

The `trainWordEmbedding` function supports `tokenizedDocument` or file input regardless of language. If you have a `tokenizedDocument` array or a file containing your data in the correct format, then you can use this function.

See Also

`addLanguageDetails` | `addPartOfSpeechDetails` | `normalizeWords` | `removeStopWords` | `stopWords` | `tokenDetails` | `tokenizedDocument`

More About

- “Language Support” on page 1-27
- “Analyze Japanese Text Data” on page 1-35

Analyze Japanese Text Data

This example shows how to import, prepare, and analyze Japanese text data using a topic model.

Japanese text data can be large and can contain lots of noise that negatively affects statistical analysis. For example, the text data can contain the following:

- Variations in word forms. For example, "難しい" ("is difficult") and "難しかった" ("was difficult")
- Words that add noise. For example, stop words such as "あそこ" ("over there"), "あたり" ("around"), and "あちら" ("there")
- Punctuation and special characters

These word clouds illustrate word frequency analysis applied to some raw text data from "吾輩は猫である" by 夏目漱石, and a preprocessed version of the same text data.

This example first shows how to import and prepare Japanese text data, and then it shows how to analyze the text data using a Latent Dirichlet Allocation (LDA) model. An LDA model is a topic model that discovers underlying topics in a collection of documents and infers the word probabilities in topics. Use these steps in preparing the text data and fitting the model:

- Read HTML code from a website.
- Parse the HTML code and extract the relevant data.
- Prepare the text data for analysis using standard preprocessing techniques.
- Fit a topic model and visualize the results.

Import Data

Read the data from "吾輩は猫である" by 夏目漱石 from https://www.aozora.gr.jp/cards/000148/files/789_14547.html using the `webread` function.

Specify the character encoding of the text using the `weboptions` function. To find the correct character encoding for a HTML, look in the header of the HTML code. For this file, specify the character encoding to be "`Shift_JIS`".

```
url = "https://www.aozora.gr.jp/cards/000148/files/789_14547.html";
options = weboptions('CharacterEncoding','Shift_JIS');
code = webread(url,options);
```

View the first few lines of the HTML code.

```
extractBefore(code,"<script")
```

```
ans =
    '<?xml version="1.0" encoding="Shift_JIS"?>
     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
         "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
     <html xml:lang="ja" >
     <head>
         <meta http-equiv="Content-Type" content="text/html;charset=Shift_JIS" />
         <meta http-equiv="content-style-type" content="text/css" />
         <link rel="stylesheet" type="text/css" href="../../aozora.css" />
         <title>夏目漱石 吾輩は猫である</title>
         '
```

Extract the text data from the HTML using `extractHTMLText`. Split the text by `newline` characters.

```
textData = extractHTMLText(code);
textData = string(split(textData,newline));
textData(1:10)
```

```
ans = 10×1 string array
    "吾輩は猫である"
    ""
    "夏目漱石"
    ""
    " "
    ""
    "―"
```

```
    ""
    " 吾輩は猫である。名前はまだ無い。"
    " どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶し
```

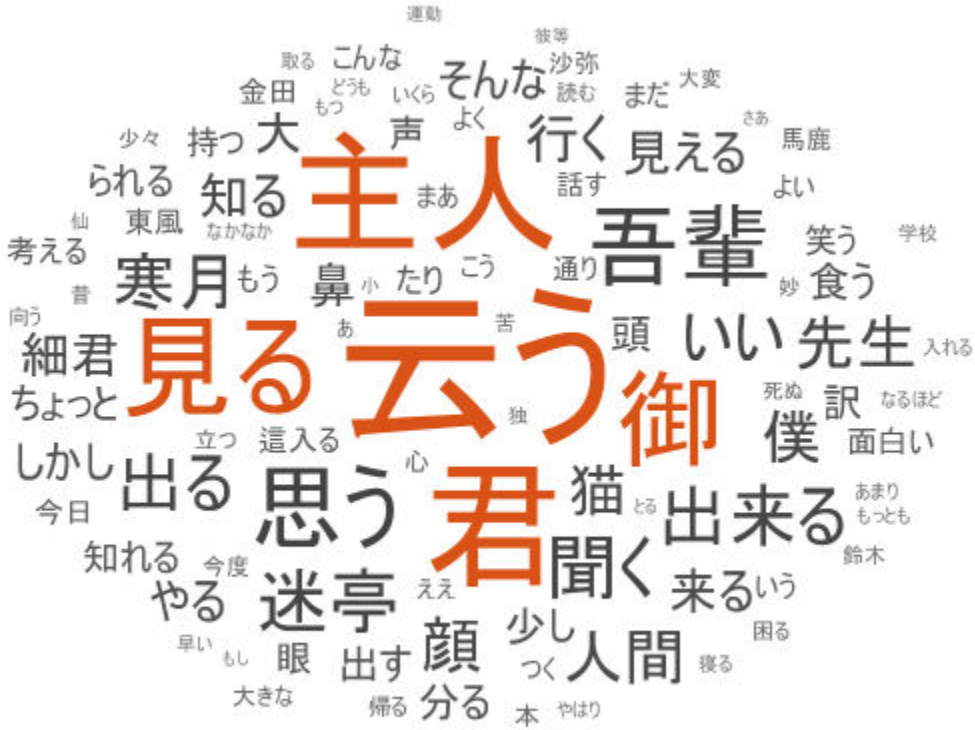
Remove the empty lines of text.

```
idx = textData == "";
textData(idx) = [];
textData(1:10)
```

```
ans = 10×1 string array
    "吾輩は猫である"
    "夏目漱石"
    " "
    "一"
    " 吾輩は猫である。名前はまだ無い。"
    " どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶し
    " この書生の掌の裏でしばらくはよい心持に坐っておったが、しばらくすると非常な速力で運転し始めた。書生
    " ふと気が付いて見ると書生はいない。たくさんおった兄弟が一疋も見えぬ。肝心の母親さえ姿を隠してしまっ
    " ようやくの思いで笹原を這い出すと向うに大きな池がある。吾輩は池の前に坐ってどうしたらよかろうと考え
    " 吾輩の主人は滅多に吾輩と顔を合せる事がない。職業は教師だそうだ。学校から帰ると終日書斎に這入ったき
```

Visualize the text data in a word cloud.

```
figure
wordcloud(textData);
```



Prepare Text Data for Analysis

Tokenize the text using `tokenizedDocument` and view the first few documents.

```
documents = tokenizedDocument(textData);
documentsRaw = documents;

documents(1:10)
```

```
ans =
  10×1 tokenizedDocument:

      5 tokens: 吾輩 は 猫 で ある
      2 tokens: 夏目 漱石
```

```
      0 tokens:
      1 tokens: 一
     11 tokens: 吾輩 は 猫 で ある 。 名前 は まだ 無い 。
    264 tokens: どこ で 生れ た か とんと 見当 が つか ぬ 。 何 でも 薄暗い じめじめ し た 所 で
    100 tokens: この 書生 の 掌 の 裏 で しばらく は よい 心持 に 坐っ て おっ た が 、 しばらく
     92 tokens: ふと 気 が 付い て 見る と 書生 は い ない 。 たくさん おっ た 兄弟 が 一疋 も 見
    693 tokens: ようやく の 思い で 笹原 を 這い出す と 向う に 大きな 池 が ある 。 吾輩 は 池 の
    276 tokens: 吾輩 の 主人 は 滅多 に 吾輩 と 顔 を 合せる 事 が ない 。 職業 は 教師 だ そう た
```

Remove the stop words.

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

```
ans =
  10×1 tokenizedDocument:

      2 tokens: 吾輩 猫
      2 tokens: 夏目 漱石
      0 tokens:
      0 tokens:
      6 tokens: 吾輩 猫 。 まだ 無い 。
    117 tokens: 生れ とんと 見当 つか ぬ 。 薄暗い じめじめ ニャーニャー 泣い いた事 記憶 。 吾輩
     43 tokens: 書生 掌 裏 しばらく よい 心持 坐っ おっ 、 しばらく 非常 速力 運転 始め 。 書生 動
     46 tokens: ふと 付い 見る 書生 。 おっ 兄弟 一疋 見え ぬ 。 肝心 母親 姿 隠し しまっ 。 上今
    323 tokens: ようやく 思い 笹原 這い出す 向う 大きな 池 。 吾輩 池 坐っ たら よかろ 考え 。 別に
    122 tokens: 吾輩 主人 滅多 吾輩 顔 合せる 。 職業 教師 。 学校 帰る 終日 書斎 這入っ ぎりほとん
```

Erase the punctuation.

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

```
ans =
  10×1 tokenizedDocument:

      2 tokens: 吾輩 猫
      2 tokens: 夏目 漱石
      0 tokens:
      0 tokens:
      4 tokens: 吾輩 猫 まだ 無い
    102 tokens: 生れ とんと 見当 つか ぬ 薄暗い じめじめ ニャーニャー 泣い いた事 記憶 吾輩 始め 人
     36 tokens: 書生 掌 裏 しばらく よい 心持 坐っ おっ しばらく 非常 速力 運転 始め 書生 動く 動く
```

```
    38 tokens: ふと 付い 見る 書生 おっ 兄弟 一疋 見え ぬ 肝心 母親 姿 隠し しまっ 上今 違っ 無
   274 tokens: ようやく 思い 笹原 這い出す 向う 大きな 池 吾輩 池 坐っ たら よかろ 考え 別に とい
   101 tokens: 吾輩 主人 滅多 吾輩 顔 合せる 職業 教師 学校 帰る 終日 書斎 這入っ ぎりほとんど 出
```

Lemmatize the text using `normalizeWords`.

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

```
ans =
  10×1 tokenizedDocument:

     2 tokens: 吾輩 猫
     2 tokens: 夏目 漱石
     0 tokens:
     0 tokens:
     4 tokens: 吾輩 猫 まだ 無い
   102 tokens: 生れる とんと 見当 つく ぬ 薄暗い じめじめ ニャーニャー 泣く いた事 記憶 吾輩 始め
    36 tokens: 書生 掌 裏 しばらく よい 心持 坐る おる しばらく 非常 速力 運転 始める 書生 動く 
    38 tokens: ふと 付く 見る 書生 おる 兄弟 一疋 見える ぬ 肝心 母親 姿 隠す しまう 上今 違う 無
   274 tokens: ようやく 思い 笹原 這い出す 向う 大きな 池 吾輩 池 坐る た よい 考える 別に という
   101 tokens: 吾輩 主人 滅多 吾輩 顔 合せる 職業 教師 学校 帰る 終日 書斎 這入る ぎりほとんど 出
```

Some preprocessing steps, such as removing stop words and erasing punctuation, return empty documents. Remove the empty documents using the `removeEmptyDocuments` function.

```
documents = removeEmptyDocuments(documents);
```

Create Preprocessing Function

Creating a function that performs preprocessing can be useful to prepare different collections of text data in the same way. For example, you can use a function to preprocess new data using the same steps as the training data.

Create a function which tokenizes and preprocesses the text data to use for analysis. The function `preprocessJapaneseText`, performs these steps:

1. Tokenize the text using `tokenizedDocument`.
2. Erase punctuation using `erasePunctuation`.
3. Remove a list of stop words (such as "あそこ", "あたり", and "あちら") using `removeStopWords`.

4 Lemmatize the words using `normalizeWords`.

Remove the empty documents after preprocessing using the `removeEmptyDocuments` function. Removing documents after using a preprocessing function makes it easier to remove corresponding data such as labels from other sources.

In this example, use the preprocessing function `preprocessJapaneseText`, listed at the end of the example, to prepare the text data.

```
documents = preprocessJapaneseText(textData);
documents(1:5)
```

```
ans =
  5×1 tokenizedDocument:

    2 tokens: 吾輩 猫
    2 tokens: 夏目 漱石
    0 tokens:
    0 tokens:
    4 tokens: 吾輩 猫 まだ 無い
```

Remove the empty documents.

```
documents = removeEmptyDocuments(documents);
```

Get Part-of-Speech Tags

Get the token details and then view the details of the first few tokens.

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

```
ans=8×7 table
    Token     DocumentNumber    LineNumber      Type      Language    PartOfSpeech    Le
    ______    ______________    __________    ________    ________    ____________    __

    "吾輩"          1               1         letters       ja       pronoun          "吾
    "猫"            1               1         letters       ja       noun             "猫
    "夏目"          2               1         letters       ja       proper-noun      "夏
    "漱石"          2               1         letters       ja       proper-noun      "漱
    "吾輩"          3               1         letters       ja       pronoun          "吾
    "猫"            3               1         letters       ja       noun             "猫
    "まだ"          3               1         letters       ja       adverb           "ま
```

```
    "無い"          3               1         letters       ja       adjective       "無
```

The `PartOfSpeech` variable in the table contains the part-of-speech tags of the tokens. Create word clouds of all the nouns and adjectives, respectively.

```
figure
idx = tdetails.PartOfSpeech == "noun";
tokens = tdetails.Token(idx);
subplot(1,2,1)
wordcloud(tokens);
title("Nouns")

idx = tdetails.PartOfSpeech == "adjective";
tokens = tdetails.Token(idx);
subplot(1,2,2)
wordcloud(tokens);
title("Adjectives")
```

Fit Topic Model

Fit a latent Dirichlet allocation (LDA) topic model to the data. An LDA model discovers underlying topics in a collection of documents and infers word probabilities in topics.

To fit an LDA model to the data, you first must create a bag-of-words model. 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. Create a bag-of-words model using `bagOfWords`.

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

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

```
bag = removeEmptyDocuments(bag);
```

Fit an LDA model with seven topics using `fitlda`. To suppress the verbose output, set `'Verbose'` to `0`.

```
numTopics = 7;
mdl = fitlda(bag,numTopics,'Verbose',0);
```

Visualize the first four topics using word clouds.

```
figure
for i = 1:4
    subplot(2,2,i)
    wordcloud(mdl,i);
    title("Topic " + i)
end
```

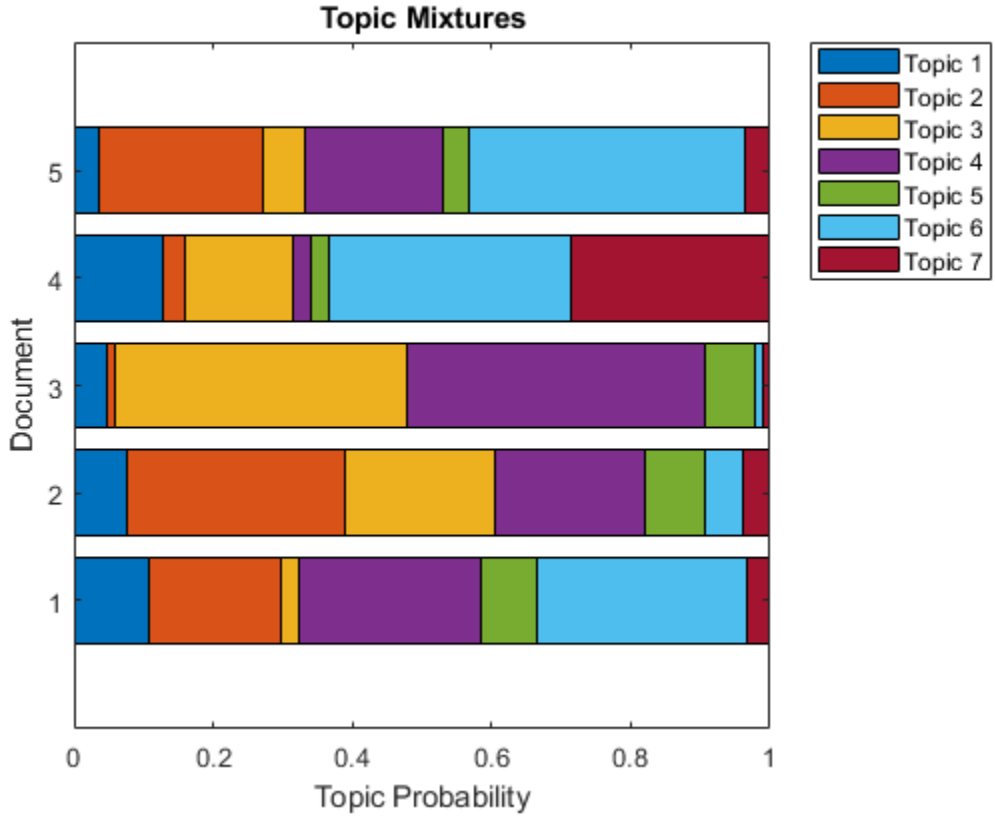
Visualize multiple topic mixtures using stacked bar charts. View five input documents at random and visualize the corresponding topic mixtures.

```
numDocuments = numel(documents);
idx = randperm(numDocuments,5);
documents(idx)
```

```
ans =
  5×1 tokenizedDocument:

    10 tokens: 馬鹿馬鹿しい じゃ いくら 唐津 掘る 来る たって 山の芋 銭 たまる
     6 tokens: ええ 少し 御 話す する 思う
    26 tokens: しかし 娘 寒月 来る 主人 鈴木 君 聞く 通り 述べる 鈴木 君 迷惑 云う 顔 付 しきりに
     8 tokens: 御 母 あさまい 夕べ 泥棒 這入る 姉 尋ねる
```

```
5 tokens: 真面目 現に 御 友達 手紙
```

```
topicMixtures = transform(mdl,documents(idx));
figure
barh(topicMixtures(1:5,:),'stacked')
xlim([0 1])
title("Topic Mixtures")
xlabel("Topic Probability")
ylabel("Document")
legend("Topic " + string(1:numTopics),'Location','northeastoutside')
```



Example Preprocessing Function

The function `preprocessJapaneseText`, performs these steps:

1. Tokenize the text using `tokenizedDocument`.
2. Erase punctuation using `erasePunctuation`.
3. Remove a list of stop words (such as "あそこ", "あたり", and "あちら") using `removeStopWords`.
4. Lemmatize the words using `normalizeWords`.

```
function documents = preprocessJapaneseText(textData)

% Tokenize the text.
documents = tokenizedDocument(textData);

% Erase the punctuation.
documents = erasePunctuation(documents);

% Remove a list of stop words.
documents = removeStopWords(documents);

% Lemmatize the words.
documents = normalizeWords(documents,'Style','lemma');
end
```

See Also

`addPartOfSpeechDetails` | `normalizeWords` | `removeStopWords` | `stopWords` | `tokenDetails` | `tokenizedDocument`

More About

- “Language Support” on page 1-27

See Also

Related Examples

- “Create Simple Text Model for Classification” on page 2-2
- “Analyze Text Data Using Topic Models” on page 2-18

- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Analyze Text Data Containing Emojis” on page 2-25
- “Train a Sentiment Classifier” on page 2-33
- “Classify Text Data Using Deep Learning” on page 2-43
- “Generate Text Using Deep Learning” (Deep Learning Toolbox)

Modeling and Prediction

- “Create Simple Text Model for Classification” on page 2-2
- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Analyze Text Data Using Topic Models” on page 2-18
- “Analyze Text Data Containing Emojis” on page 2-25
- “Train a Sentiment Classifier” on page 2-33
- “Classify Text Data Using Deep Learning” on page 2-43
- “Pride and Prejudice and MATLAB” on page 2-55
- “Word-By-Word Text Generation Using Deep Learning” on page 2-62
- “Classify Out-of-Memory Text Data Using Custom Mini-Batch Datastore” on page 2-69
- “Choose Number of Topics for LDA Model” on page 2-75
- “Compare LDA Solvers” on page 2-80

Create Simple Text Model for Classification

This example shows how to train a simple text classifier on word frequency counts using a bag-of-words model.

You can create a simple classification model which uses word frequency counts as predictors. This example trains a simple classification model to predict the event type of weather reports using text descriptions.

To reproduce the results of this example, set `rng` to `'default'`.

```
rng('default')
```

Load and Extract Text Data

Load the example data. The file `weatherReports.csv` contains weather reports, including a text description and categorical labels for each event.

```
filename = "weatherReports.csv";
data = readtable(filename,'TextType','string');
head(data)
```

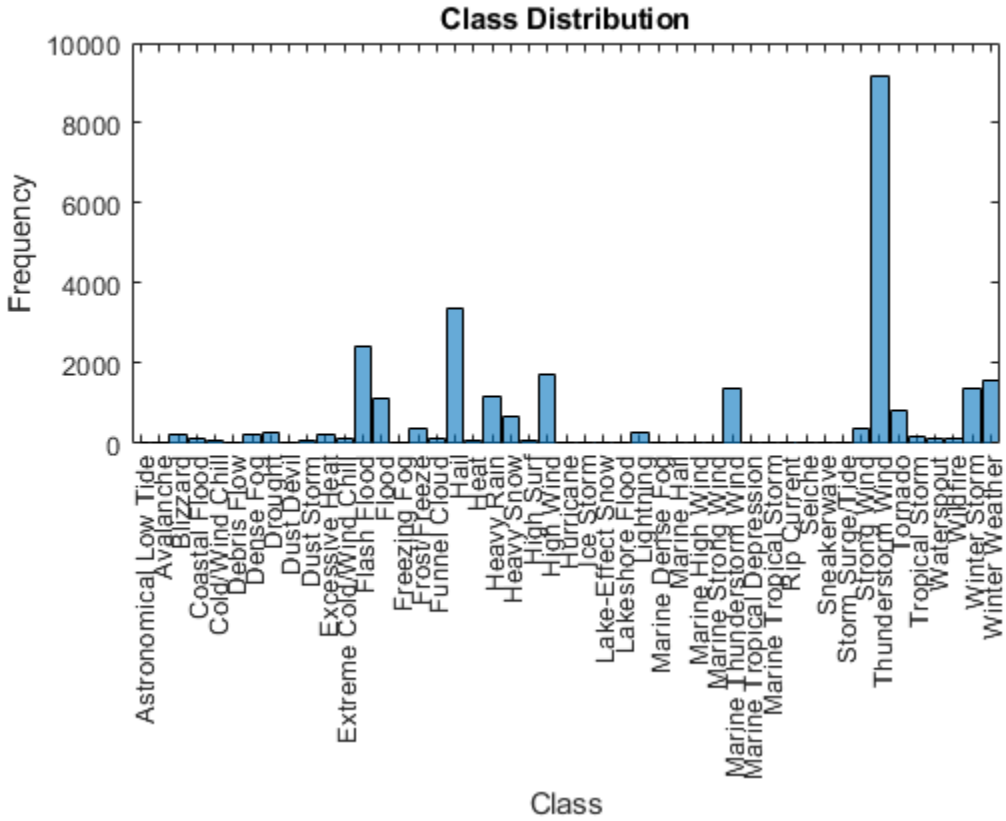
```
ans=8×16 table
            Time             event_id          state              event_type          da
    _____________________    _________    _______________    ____________________    __

    22-Jul-2016 16:10:00    6.4433e+05    "MISSISSIPPI"      "Thunderstorm Wind"
    15-Jul-2016 17:15:00    6.5182e+05    "SOUTH CAROLINA"   "Heavy Rain"
    15-Jul-2016 17:25:00    6.5183e+05    "SOUTH CAROLINA"   "Thunderstorm Wind"
    16-Jul-2016 12:46:00    6.5183e+05    "NORTH CAROLINA"   "Thunderstorm Wind"
    15-Jul-2016 14:28:00    6.4332e+05    "MISSOURI"         "Hail"
    15-Jul-2016 16:31:00    6.4332e+05    "ARKANSAS"         "Thunderstorm Wind"
    15-Jul-2016 16:03:00    6.4343e+05    "TENNESSEE"        "Thunderstorm Wind"
    15-Jul-2016 17:27:00    6.4344e+05    "TENNESSEE"        "Hail"
```

Remove rows with empty reports.

```
idx = strlength(data.event_narrative) == 0;
data(idx,:) = [];
```

Convert the labels in the `event_type` column of the table to categorical and view the distribution of the classes in the data using a histogram.

```
data.event_type = categorical(data.event_type);
figure
h = histogram(data.event_type);
xlabel("Class")
ylabel("Frequency")
title("Class Distribution")
```



The classes of the data are imbalanced, with several classes containing few observations. To ensure that you can partition the data so that the partitions contain observations for each class, remove any classes which appear fewer than ten times.

Get the frequency counts of the classes and their names from the histogram.

```
classCounts = h.BinCounts;
classNames = h.Categories;
```

Find the classes containing fewer than ten observations and remove these infrequent classes from the data.

```
idxLowCounts = classCounts < 10;
infrequentClasses = classNames(idxLowCounts);
idxInfrequent = ismember(data.event_type,infrequentClasses);
data(idxInfrequent,:) = [];
```

Partition the data into a training partition and a held-out test set. Specify the holdout percentage to be 10%.

```
cvp = cvpartition(data.event_type,'Holdout',0.1);
dataTrain = data(cvp.training,:);
dataTest = data(cvp.test,:);
```

Extract the text data and labels from the tables.

```
textDataTrain = dataTrain.event_narrative;
textDataTest = dataTest.event_narrative;
YTrain = dataTrain.event_type;
YTest = dataTest.event_type;
```

Prepare Text Data for Analysis

Create a function which tokenizes and preprocesses the text data so it can be used for analysis. The function `preprocessWeatherNarratives`, performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Lemmatize the words using `normalizeWords`.

Use the example preprocessing function `preprocessWeatherNarratives` to prepare the text data.

```
documents = preprocessWeatherNarratives(textDataTrain);
documents(1:5)

ans =
  5×1 tokenizedDocument:

   (1,1)   5 tokens: large tree down plantersville nettleton
   (2,1)  18 tokens: two foot deep standing water develop street winthrop unive…
   (3,1)   9 tokens: nws columbia relayed report tree blow down tom hall
   (4,1)  10 tokens: medium report two tree blow down i40 old fort area
   (5,1)   8 tokens: few tree limb great inches down hwy roseland
```

Create a bag-of-words model from the tokenized documents.

```
bag = bagOfWords(documents)

bag =
  bagOfWords with properties:

          Counts: [25316×17458 double]
      Vocabulary: [1×17458 string]
        NumWords: 17458
    NumDocuments: 25316
```

Remove words from the bag-of-words model that do not appear more than two times in total. Remove any documents containing no words from the bag-of-words model, and remove the corresponding entries in labels.

```
bag = removeInfrequentWords(bag,2);
[bag,idx] = removeEmptyDocuments(bag);
YTrain(idx) = [];
bag

bag =
  bagOfWords with properties:

          Counts: [25315×6515 double]
      Vocabulary: [1×6515 string]
        NumWords: 6515
    NumDocuments: 25315
```

Train Supervised Classifier

Train a supervised classification model using the word frequency counts from the bag-of-words model and the labels.

Train a multiclass linear classification model using `fitcecoc`. Specify the `Counts` property of the bag-of-words model to be the predictors, and the event type labels to be the response. Specify the learners to be linear. These learners support sparse data input.

```
XTrain = bag.Counts;
mdl = fitcecoc(XTrain,YTrain,'Learners','linear')
```

```
mdl =
  classreg.learning.classif.CompactClassificationECOC
      ResponseName: 'Y'
        ClassNames: [1×39 categorical]
    ScoreTransform: 'none'
    BinaryLearners: {741×1 cell}
      CodingMatrix: [39×741 double]


  Properties, Methods
```

For a better fit, you can try specifying different parameters of the linear learners. For more information on linear classification learner templates, see `templateLinear`.

Test Classifier

Predict the labels of the test data using the trained model and calculate the classification accuracy. The classification accuracy is the proportion of the labels that the model predicts correctly.

Preprocess the test data using the same preprocessing steps as the training data. Encode the resulting test documents as a matrix of word frequency counts according to the bag-of-words model.

```
documentsTest = preprocessWeatherNarratives(textDataTest);
XTest = encode(bag,documentsTest);
```

Predict the labels of the test data using the trained model and calculate the classification accuracy.

```
YPred = predict(mdl,XTest);
acc = sum(YPred == YTest)/numel(YTest)
```

```
acc = 0.8729
```

Predict Using New Data

Classify the event type of new weather reports. Create a string array containing the new weather reports.

```
str = [ ...
    "A large tree is downed and blocking traffic outside Apple Hill."
    "Damage to many car windshields in parking lot."
    "Lots of water damage to computer equipment inside the office."];
documentsNew = preprocessWeatherNarratives(str);
XNew = encode(bag,documentsNew);
labelsNew = predict(mdl,XNew)
```

```
labelsNew = 3×1 categorical array
     Thunderstorm Wind
     Thunderstorm Wind
     Flash Flood
```

Example Preprocessing Function

The function `preprocessWeatherNarratives`, performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Lemmatize the words using `normalizeWords`.

```
function documents = preprocessWeatherNarratives(textData)
% Convert the text data to lowercase.
cleanTextData = lower(textData);

% Tokenize the text.
documents = tokenizedDocument(cleanTextData);

% Erase punctuation.
documents = erasePunctuation(documents);
```

```
% Remove a list of stop words.
documents = removeStopWords(documents);

% Remove words with 2 or fewer characters, and words with 15 or greater
% characters.
documents = removeShortWords(documents,2);
documents = removeLongWords(documents,15);

% Lemmatize the words.
documents = addPartOfSpeechDetails(documents);
documents = normalizeWords(documents,'Style','lemma');

end
```

See Also

`addPartOfSpeechDetails` | `bagOfWords` | `encode` | `erasePunctuation` | `normalizeWords` | `removeLongWords` | `removeShortWords` | `removeStopWords` | `tokenizedDocument` | `wordcloud`

Related Examples

- “Analyze Text Data Using Topic Models” on page 2-18
- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Analyze Text Data Containing Emojis” on page 2-25
- “Train a Sentiment Classifier” on page 2-33
- “Classify Text Data Using Deep Learning” on page 2-43
- “Generate Text Using Deep Learning” (Deep Learning Toolbox)

Analyze Text Data Using Multiword Phrases

This example shows how to analyze text using n-gram frequency counts.

N-Grams

An n-gram is a tuple of n consecutive words. For example, a bigram (the case when $n = 2$) is a pair of consecutive words such as "heavy rainfall". A unigram (the case when $n = 1$) is a single word. A bag-of-n-grams model records the number of times that different n-grams appear in document collections.

Using a bag-of-n-grams model, you can retain more information on word ordering in the original text data. For example, a bag-of-n-grams model is better suited for capturing short phrases which appear in the text, such as "heavy rainfall" and "thunderstorm winds".

To create a bag-of-n-grams model, use `bagOfNgrams`. You can input `bagOfNgrams` objects into other Text Analytics Toolbox functions such as `wordcloud` and `fitlda`.

Load and Extract Text Data

To reproduce the results of this example, set `rng` to `'default'`.

```
rng('default')
```

Load the example data. The file `weatherReports.csv` contains weather reports, including a text description and categorical labels for each event. Remove the rows with empty reports.

```
filename = "weatherReports.csv";
data = readtable(filename,'TextType','String');
idx = strlength(data.event_narrative) == 0;
data(idx,:) = [];
```

Extract the text data from the table and view the first few reports.

```
textData = data.event_narrative;
textData(1:5)
```

```
ans = 5×1 string array
    "Large tree down between Plantersville and Nettleton."
    "One to two feet of deep standing water developed on a street on the Winthrop Unive
```

```
"NWS Columbia relayed a report of trees blown down along Tom Hall St."
"Media reported two trees blown down along I-40 in the Old Fort area."
"A few tree limbs greater than 6 inches down on HWY 18 in Roseland."
```

Prepare Text Data for Analysis

Create a function which tokenizes and preprocesses the text data so it can be used for analysis. The function `preprocessWeatherNarratives` listed at the end of the example, performs the following steps:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Lemmatize the words using `normalizeWords`.

Use the example preprocessing function `preprocessWeatherNarratives` to prepare the text data.

```
documents = preprocessWeatherNarratives(textData);
documents(1:5)
```

```
ans =
  5×1 tokenizedDocument:

   (1,1)   5 tokens: large tree down plantersville nettleton
   (2,1)  18 tokens: two foot deep standing water develop street winthrop unive…
   (3,1)   9 tokens: nws columbia relayed report tree blow down tom hall
   (4,1)  10 tokens: medium report two tree blow down i40 old fort area
   (5,1)   8 tokens: few tree limb great inches down hwy roseland
```

Create Word Cloud of Bigrams

Create a word cloud of bigrams by first creating a bag-of-n-grams model using `bagOfNgrams`, and then inputting the model to `wordcloud`.

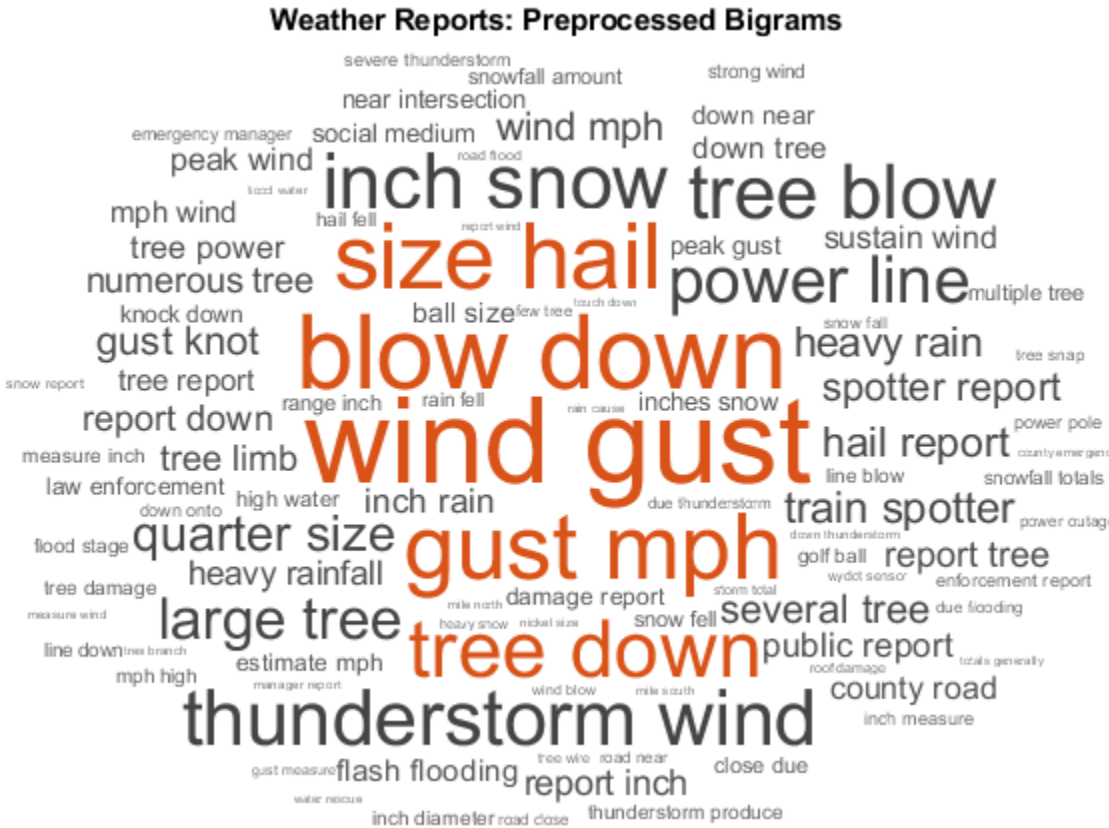
To count the n-grams of length 2 (bigrams), use `bagOfNgrams` with the default options.

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

```
bag =
  bagOfNgrams with properties:

          Counts: [28138×117043 double]
      Vocabulary: [1×18409 string]
          Ngrams: [117043×2 string]
    NgramLengths: 2
       NumNgrams: 117043
    NumDocuments: 28138
```

Visualize the bag-of-n-grams model using a word cloud.

```
figure
wordcloud(bag);
title("Weather Reports: Preprocessed Bigrams")
```



Fit Topic Model to Bag-of-N-Grams

A Latent Dirichlet Allocation (LDA) model is a topic model which discovers underlying topics in a collection of documents and infers the word probabilities in topics.

Create an LDA topic model with 10 topics using `fitlda`. The function fits an LDA model by treating the n-grams as single words.

```
mdl = fitlda(bag,10);
```

```
Initial topic assignments sampled in 0.741989 seconds.
=================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
```

```
|            | (seconds)  |   log(L)   |           |               |   iterations  |
=============================================================================================
|          0 |       2.81 |            |  2.043e+04 |         2.500 |             0 |
|          1 |       3.62 | 6.8345e-02 |  1.083e+04 |         2.500 |             0 |
|          2 |       3.54 | 1.9129e-03 |  1.064e+04 |         2.500 |             0 |
|          3 |       3.79 | 2.4671e-04 |  1.061e+04 |         2.500 |             0 |
|          4 |       3.81 | 8.5912e-05 |  1.060e+04 |         2.500 |             0 |
=============================================================================================
```

Visualize the first four topics as word clouds.

```
figure
for i = 1:4
    subplot(2,2,i)
    wordcloud(mdl,i);
    title("LDA Topic " + i)
end
```

The word clouds highlight commonly co-occuring bigrams in the LDA topics. The function plots the bigrams with sizes according to their probabilities for the specified LDA topics.

Analyze Text Using Longer Phrases

To analyze text using longer phrases, specify the `'NGramLengths'` option in `bagOfNgrams` to be a larger value.

When working with longer phrases, it can be useful to keep stop words in the model. For example, to detect the phrase "is not happy", keep the stop words "is" and "not" in the model.

Preprocess the text. Erase the punctuation using `erasePunctuation`, and tokenize using `tokenizedDocument`.

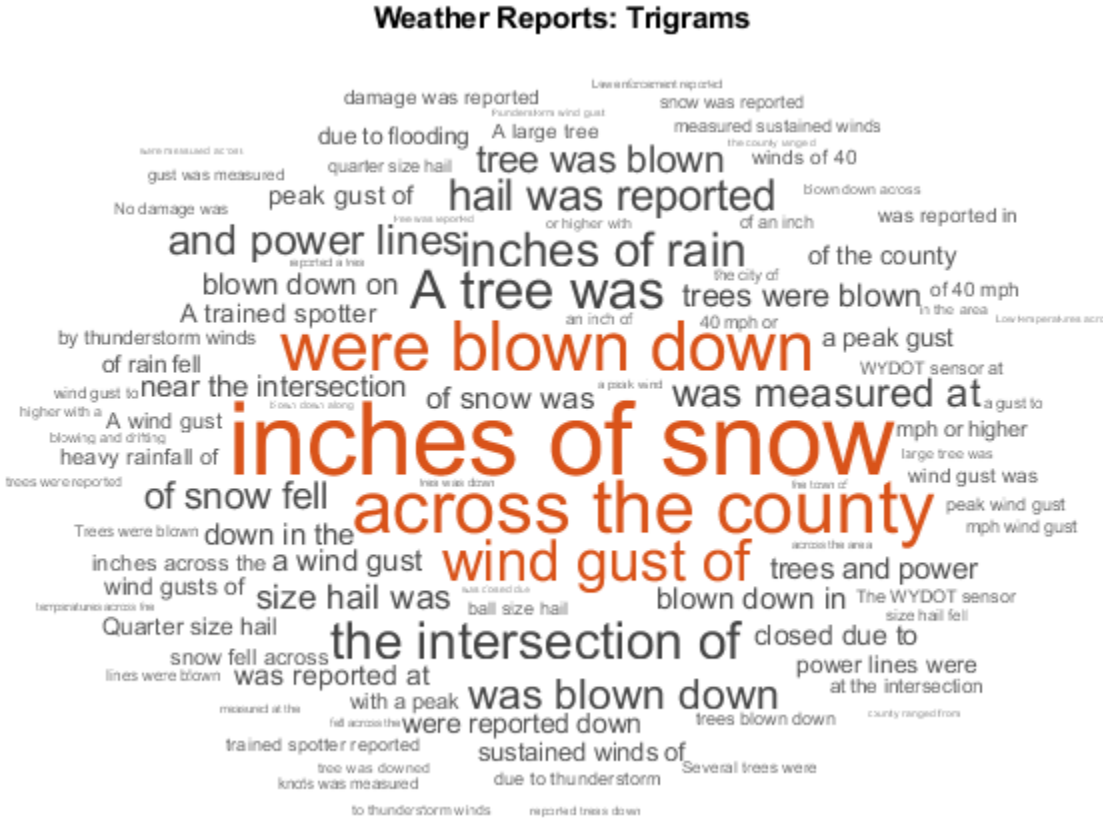
```
cleanTextData = erasePunctuation(textData);
documents = tokenizedDocument(cleanTextData);
```

To count the n-grams of length 3 (trigrams), use `bagOfNgrams` and specify `'NGramLengths'` to be 3.

```
bag = bagOfNgrams(documents,'NGramLengths',3);
```

Visualize the bag-of-n-grams model using a word cloud. The word cloud of trigrams better shows the context of the individual words.

```
figure
wordcloud(bag);
title("Weather Reports: Trigrams")
```



View the top 10 trigrams and their frequency counts using `topkngrams`.

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

```
tbl=10×3 table
                      Ngram                       Count    NgramLength
    ___________________________________________    _____    ___________

    "inches"    "of"              "snow"           2075          3
    "across"    "the"             "county"         1318          3
    "were"      "blown"           "down"           1189          3
    "wind"      "gust"            "of"              934          3
    "A"         "tree"            "was"             860          3
    "the"       "intersection"    "of"              812          3
    "inches"    "of"              "rain"            739          3
    "hail"      "was"             "reported"        648          3
    "was"       "blown"           "down"            638          3
    "and"       "power"           "lines"           631          3
```

Example Preprocessing Function

The function `preprocessWeatherNarratives` performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Lemmatize the words using `normalizeWords`.

```
function [documents] = preprocessWeatherNarratives(textData)

% Convert the text data to lowercase.
cleanTextData = lower(textData);

% Tokenize the text.
documents = tokenizedDocument(cleanTextData);

% Erase punctuation.
documents = erasePunctuation(documents);
```

```
% Remove a list of stop words.
documents = removeStopWords(documents);

% Remove words with 2 or fewer characters, and words with 15 or greater
% characters.
documents = removeShortWords(documents,2);
documents = removeLongWords(documents,15);

% Lemmatize the words.
documents = addPartOfSpeechDetails(documents);
documents = normalizeWords(documents,'Style','lemma');
end
```

See Also

`addPartOfSpeechDetails` | `bagOfNgrams` | `bagOfWords` | `erasePunctuation` | `fitlda` | `ldaModel` | `normalizeWords` | `removeLongWords` | `removeShortWords` | `removeStopWords` | `tokenizedDocument` | `topkngrams` | `wordcloud`

Related Examples

- “Create Simple Text Model for Classification” on page 2-2
- “Analyze Text Data Containing Emojis” on page 2-25
- “Analyze Text Data Using Topic Models” on page 2-18
- “Train a Sentiment Classifier” on page 2-33
- “Classify Text Data Using Deep Learning” on page 2-43
- “Generate Text Using Deep Learning” (Deep Learning Toolbox)

Analyze Text Data Using Topic Models

This example shows how to use the Latent Dirichlet Allocation (LDA) topic model to analyze text data.

A Latent Dirichlet Allocation (LDA) model is a topic model which discovers underlying topics in a collection of documents and infers the word probabilities in topics.

To reproduce the results of this example, set `rng` to `'default'`.

```
rng('default')
```

Load and Extract Text Data

Load the example data. The file `weatherReports.csv` contains weather reports, including a text description and categorical labels for each event.

```
data = readtable("weatherReports.csv",'TextType','string');
head(data)
```

```
ans=8×16 table
            Time             event_id          state              event_type          da
    _____________________    _________    _______________    ____________________    __

    22-Jul-2016 16:10:00    6.4433e+05    "MISSISSIPPI"       "Thunderstorm Wind"
    15-Jul-2016 17:15:00    6.5182e+05    "SOUTH CAROLINA"    "Heavy Rain"
    15-Jul-2016 17:25:00    6.5183e+05    "SOUTH CAROLINA"    "Thunderstorm Wind"
    16-Jul-2016 12:46:00    6.5183e+05    "NORTH CAROLINA"    "Thunderstorm Wind"
    15-Jul-2016 14:28:00    6.4332e+05    "MISSOURI"          "Hail"
    15-Jul-2016 16:31:00    6.4332e+05    "ARKANSAS"          "Thunderstorm Wind"
    15-Jul-2016 16:03:00    6.4343e+05    "TENNESSEE"         "Thunderstorm Wind"
    15-Jul-2016 17:27:00    6.4344e+05    "TENNESSEE"         "Hail"
```

Extract the text data from the field `event_narrative`.

```
textData = data.event_narrative;
textData(1:10)
```

```
ans = 10×1 string array
    "Large tree down between Plantersville and Nettleton."
    "One to two feet of deep standing water developed on a street on the Winthrop Unive
    "NWS Columbia relayed a report of trees blown down along Tom Hall St."
    "Media reported two trees blown down along I-40 in the Old Fort area."
```

```
    ""
    "A few tree limbs greater than 6 inches down on HWY 18 in Roseland."
    "Awning blown off a building on Lamar Avenue. Multiple trees down near the intersec
    "Quarter size hail near Rosemark."
    "Tin roof ripped off house on Old Memphis Road near Billings Drive. Several large t
    "Powerlines down at Walnut Grove and Cherry Lane roads."
```

Prepare Text Data for Analysis

Create a function which tokenizes and preprocesses the text data so it can be used for analysis. The function `preprocessWeatherNarratives`, performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Lemmatize the words using `normalizeWords`.

Use the example preprocessing function `preprocessWeatherNarratives` to prepare the text data.

```
documents = preprocessWeatherNarratives(textData);
documents(1:5)
```

```
ans =
  5×1 tokenizedDocument:

   (1,1)   5 tokens: large tree down plantersville nettleton
   (2,1)  18 tokens: two foot deep standing water develop street winthrop unive…
   (3,1)   9 tokens: nws columbia relayed report tree blow down tom hall
   (4,1)  10 tokens: medium report two tree blow down i40 old fort area
   (5,1)   0 tokens:
```

Create a bag-of-words model from the tokenized documents.

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

```
bag =
  bagOfWords with properties:

         Counts: [36176×18410 double]
      Vocabulary: [1×18410 string]
        NumWords: 18410
    NumDocuments: 36176
```

Remove words from the bag-of-words model that have do not appear more than two times in total. Remove any documents containing no words from the bag-of-words model.

```
bag = removeInfrequentWords(bag,2);
bag = removeEmptyDocuments(bag)
```

```
bag =
  bagOfWords with properties:

         Counts: [28137×6952 double]
      Vocabulary: [1×6952 string]
        NumWords: 6952
    NumDocuments: 28137
```

Fit LDA Model

Fit an LDA model with 7 topics. For an example showing how to choose the number of topics, see “Choose Number of Topics for LDA Model” on page 2-75. To suppress verbose output, set `'Verbose'` to 0.

```
numTopics = 7;
mdl = fitlda(bag,numTopics,'Verbose',0);
```

If you have a large dataset, then the stochastic approximate variational Bayes solver is usually better suited as it can fit a good model in fewer passes of the data. The default solver for `fitlda` (collapsed Gibbs sampling) can be more accurate at the cost of taking longer to run. To use stochastic approximate variational Bayes, set the `'Solver'` option to `'savb'`. For an example showing how to compare LDA solvers, see “Analyze Text Data Using Topic Models” on page 2-18.

Visualize Topics Using Word Clouds

You can use word clouds to view the words with the highest probabilities in each topic. Visualize the first four topics using word clouds.

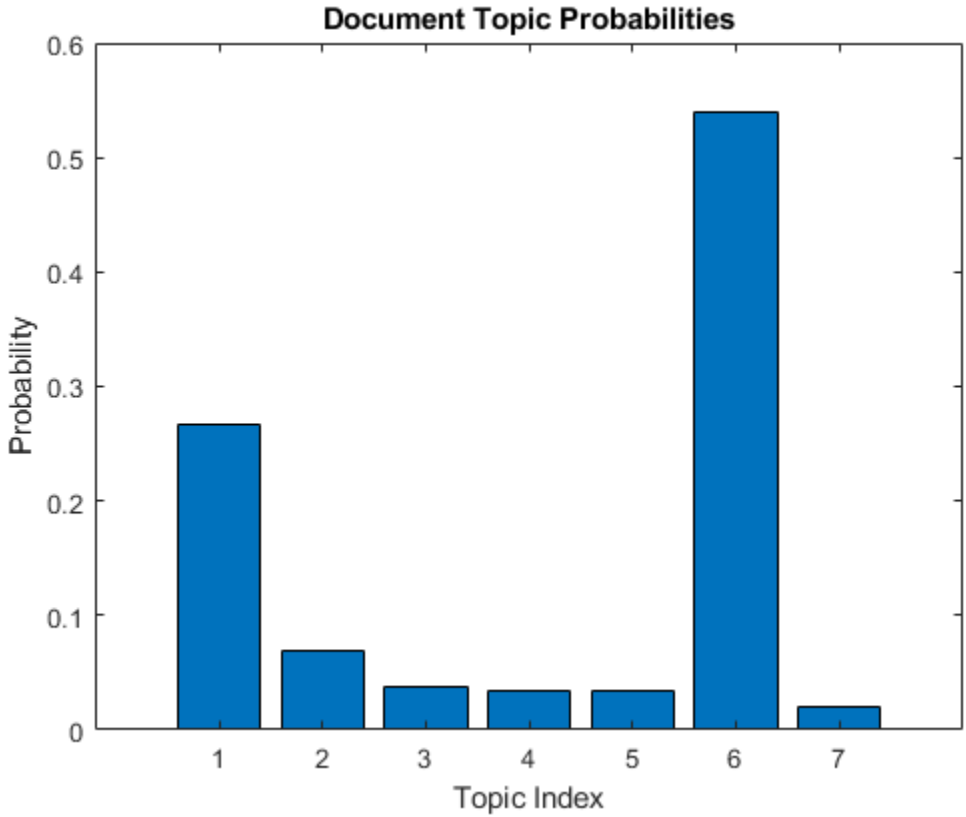
```
figure;
for topicIdx = 1:4
    subplot(2,2,topicIdx)
    wordcloud(mdl,topicIdx);
    title("Topic " + topicIdx)
end
```

View Mixtures of Topics in Documents

Use `transform` to transform the documents into vectors of topic probabilities.

```
newDocument = tokenizedDocument("A tree is downed outside Apple Hill Drive, Natick");
topicMixture = transform(mdl,newDocument);
figure
```

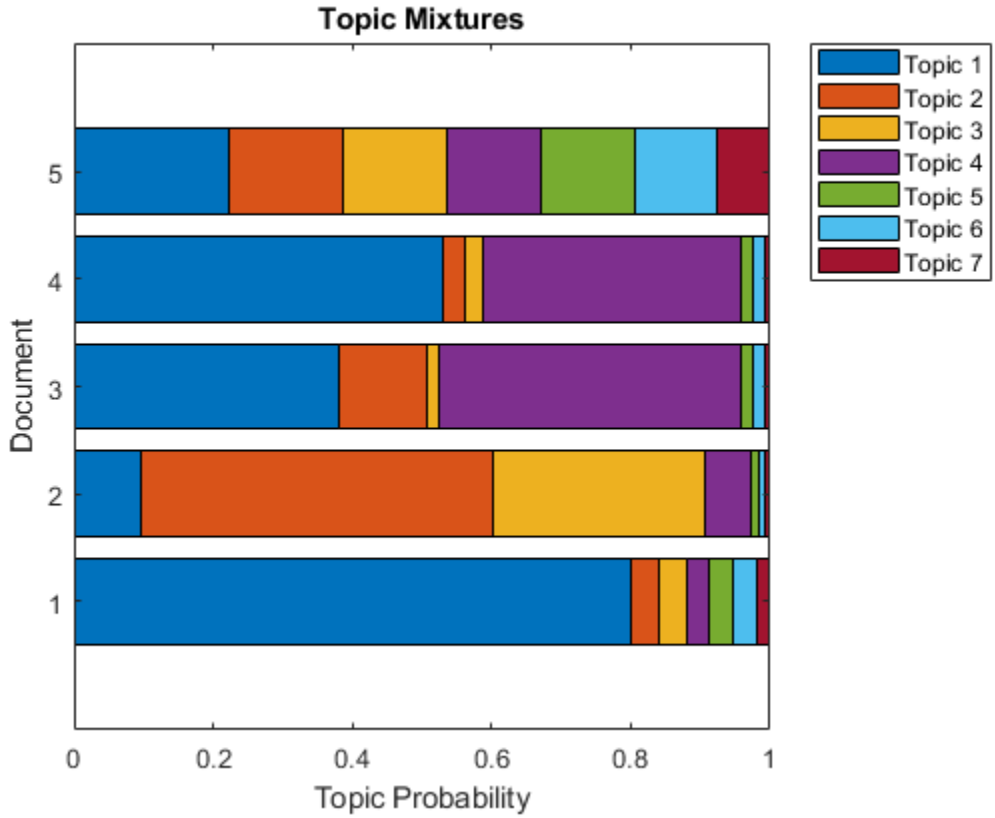
```
bar(topicMixture)
xlabel("Topic Index")
ylabel("Probability")
title("Document Topic Probabilities")
```



Visualize multiple topic mixtures using stacked bar charts. Visualize the topic mixtures of the first 5 input documents.

```
figure
topicMixtures = transform(mdl,documents(1:5));
barh(topicMixtures(1:5,:),'stacked')
xlim([0 1])
title("Topic Mixtures")
xlabel("Topic Probability")
```

```
ylabel("Document")
legend("Topic " + string(1:numTopics),'Location','northeastoutside')
```



Example Preprocessing Function

The function `preprocessWeatherNarratives`, performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeStopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.

6 Remove words with 15 or more characters using `removeLongWords`.

7 Lemmatize the words using `normalizeWords`.

```
function [documents] = preprocessWeatherNarratives(textData)
% Convert the text data to lowercase.
cleanTextData = lower(textData);

% Tokenize the text.
documents = tokenizedDocument(cleanTextData);

% Erase punctuation.
documents = erasePunctuation(documents);

% Remove a list of stop words.
documents = removeStopWords(documents);

% Remove words with 2 or fewer characters, and words with 15 or greater
% characters.
documents = removeShortWords(documents,2);
documents = removeLongWords(documents,15);

% Lemmatize the words.
documents = addPartOfSpeechDetails(documents);
documents = normalizeWords(documents,'Style','lemma');
end
```

See Also

`addPartOfSpeechDetails` | `bagOfWords` | `fitlda` | `ldaModel` | `removeEmptyDocuments` | `removeInfrequentWords` | `removeStopWords` | `tokenizedDocument` | `transform` | `wordcloud`

Related Examples

- “Create Simple Text Model for Classification” on page 2-2
- “Analyze Text Data Containing Emojis” on page 2-25
- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Train a Sentiment Classifier” on page 2-33
- “Classify Text Data Using Deep Learning” on page 2-43
- “Generate Text Using Deep Learning” (Deep Learning Toolbox)

Analyze Text Data Containing Emojis

This example shows how to analyze text data containing emojis.

Emojis are pictorial symbols that appear inline in text. When writing text on mobile devices such as smartphones and tablets, people use emojis to keep the text short and convey emotion and feelings.

You also can use emojis to analyze text data. For example, use them to identify relevant strings of text or to visualize the sentiment or emotion of the text.

When working with text data, emojis can behave unpredictably. Depending on your system fonts, your system might not display some emojis correctly. Therefore, if an emoji is not displayed correctly, then the data is not necessarily missing. Your system might be unable to display the emoji in the current font.

Composing Emojis

In most cases, you can read emojis from a file (for example, by using `extractFileText`, `extractHTMLText`, or `readtable`) or by copying and pasting them directly into MATLAB®. Otherwise, you must compose the emoji using Unicode UTF16 code units.

Some emojis consist of multiple Unicode UTF16 code units. For example, the "smiling face with sunglasses" emoji (😎 with code point U+1F60E) is a single glyph but comprises two UTF16 code units `"D83D"` and `"DE0E"`. Create a string containing this emoji using the `compose` function, and specify the two code units with the prefix `"\x"`.

```
emoji = compose("\xD83D\xDE0E")
```

```
emoji =
"😎"
```

First get the Unicode UTF16 code units of an emoji. Use `char` to get the numeric representation of the emoji, and then use `dec2hex` to get the corresponding hex value.

```
codeUnits = dec2hex(char(emoji))
```

```
codeUnits = 2x4 char array
    'D83D'
    'DE0E'
```

Reconstruct the composition string using the `strjoin` function with the empty delimiter `""`.

```
formatSpec = strjoin("\x" + codeUnits,"")
```

```
formatSpec =
"\xD83D\xDE0E"
```

```
emoji = compose(formatSpec)
```

```
emoji =
"😎"
```

Import Text Data

Extract the text data in the file `weekendUpdates.xlsx` using `readtable`. The file `weekendUpdates.xlsx` contains status updates containing the hashtags `"#weekend"` and `"#vacation"`.

```
filename = "weekendUpdates.xlsx";
tbl = readtable(filename,'TextType','string');
head(tbl)
```

```
ans=8×2 table
    ID                                          TextData
    __    ____________________________________________________________________________________

    1     "Happy anniversary! ❤ Next stop: Paris! ✈ #vacation"
    2     "Haha, BBQ on the beach, engage smug mode! 😎 😎 ❤ 😎 #vacation"
    3     "getting ready for Saturday night 😎 #yum #weekend 😎"
    4     "Say it with me - I NEED A #VACATION!!! 😩"
    5     "😎 Chilling 😎 at home for the first time in ages…This is the life! 😎 #weekend
    6     "My last #weekend before the exam 😎 😎."
    7     "can't believe my #vacation is over 😩 so unfair"
    8     "Can't wait for tennis this #weekend 😁😁😁 😎"
```

Extract the text data from the field `TextData` and view the first few status updates.

```
textData = tbl.TextData;
textData(1:5)
```

```
ans = 5x1 string array
    "Happy anniversary! ❤ Next stop: Paris! ✈ #vacation"
    "Haha, BBQ on the beach, engage smug mode! 😎 😎 ❤ 😎 #vacation"
    "getting ready for Saturday night 😎 #yum #weekend 😎"
```

```
    "Say it with me - I NEED A #VACATION!!! 😡"
    "🌴 Chilling 🍹 at home for the first time in ages…This is the life! 🏠 #weekend"
```

Visualize the text data in a word cloud.

```
figure
wordcloud(textData);
```

Filter Text Data by Emoji

Identify the status updates containing a particular emoji using the `contains` function. Find the indices of the documents containing the "smiling face with sunglasses" emoji (😎

with code U+1F60E). This emoji comprises the two Unicode UTF16 code units `"D83D"` and `"DE0E"`.

```
emoji = compose("\xD83D\xDE0E");
idx = contains(textData,emoji);
textDataSunglasses = textData(idx);
textDataSunglasses(1:5)
```

```
ans = 5x1 string array
    "Haha, BBQ on the beach, engage smug mode! □ □ ❤ □ #vacation"
    "getting ready for Saturday night □ #yum #weekend □"
    "□ Chilling □ at home for the first time in ages…This is the life! □ #weekend"
    "□ Check the out-of-office crew, we are officially ON #VACATION!! □"
    "Who needs a #vacation when the weather is this good ☀ □"
```

Visualize the extracted text data in a word cloud.

```
figure
wordcloud(textDataSunglasses);
```

Extract and Visualize Emojis

Visualize all the emojis in text data using a word cloud.

Extract the emojis. First tokenize the text using `tokenizedDocument`, and then view the first few documents.

```
documents = tokenizedDocument(textData);
documents(1:5)
```

```
ans =
  5x1 tokenizedDocument:

    11 tokens: Happy anniversary ! ❤ Next stop : Paris ! ✈ #vacation
```

```
    16 tokens: Haha , BBQ on the beach , engage smug mode ! □ □ ❤ □ #vacation
     9 tokens: getting ready for Saturday night □ #yum #weekend □
    13 tokens: Say it with me - I NEED A #VACATION ! ! ! ☹
    19 tokens: □ Chilling □ at home for the first time in ages … This is the life ! □ #
```

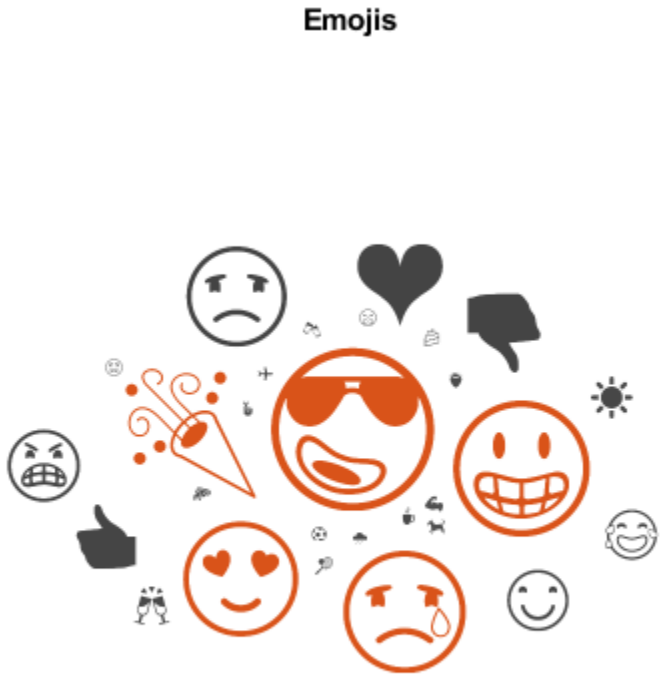
The `tokenizedDocument` function automatically detects emoji and assigns the token type `"emoji"`. View the first few token details of the documents using the `tokenDetails` function.

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

```
ans=8×5 table
        Token        DocumentNumber    LineNumber       Type        Language
    _____________    ______________    __________    ___________    ________

    "Happy"                1               1         letters          en
    "anniversary"          1               1         letters          en
    "!"                    1               1         punctuation      en
    "❤"                    1               1         emoji            en
    "Next"                 1               1         letters          en
    "stop"                 1               1         letters          en
    ":"                    1               1         punctuation      en
    "Paris"                1               1         letters          en
```

Visualize the emojis in a word cloud by extracting the tokens with token type `"emoji"` and inputting them into the `wordcloud` function.

```
idx = tdetails.Type == "emoji";
tokens = tdetails.Token(idx);
figure
wordcloud(tokens);
title("Emojis")
```



See Also

`tokenDetails` | `tokenizedDocument` | `wordcloud`

Related Examples

- “Analyze Text Data Using Topic Models” on page 2-18
- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Train a Sentiment Classifier” on page 2-33
- “Classify Text Data Using Deep Learning” on page 2-43

- “Generate Text Using Deep Learning” (Deep Learning Toolbox)

Train a Sentiment Classifier

This example shows how to train a classifier for sentiment analysis using an annotated list of positive and negative sentiment words and a pretrained word embedding.

The pretrained word embedding plays several roles in this workflow. It converts words into numeric vectors and forms the basis for a classifier. You can then use the classifier to predict the sentiment of other words using their vector representation, and use these classifications to calculate the sentiment of a piece of text. There are four steps in training and using the sentiment classifier:

- Load a pretrained word embedding.
- Load an opinion lexicon listing positive and negative words.
- Train a sentiment classifier using the word vectors of the positive and negative words.
- Calculate the mean sentiment scores of the words in a piece of text.

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load Pretrained Word Embedding

Word embeddings map words in a vocabulary to numeric vectors. These embeddings can capture semantic details of the words so that similar words have similar vectors. They also model relationships between words through vector arithmetic. For example, the relationship *king is to queen as man is to woman* is described by the equation *king – man + woman = queen.*

Load a pretrained word embedding using the `fastTextWordEmbedding` function. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding;
```

Load Opinion Lexicon

Load the positive and negative words from the opinion lexicon (also known as a sentiment lexicon) from https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html. First, extract the files from the `.rar` file into a folder named `opinion-lexicon-English`, and then import the text.

Load the data using the function `readLexicon` listed at the end of this example. The output `data` is a table with variables `Word` containing the words, and `Label` containing a categorical sentiment label, `Positive` or `Negative`.

```
data = readLexicon;
```

View the first few words labeled as positive.

```
idx = data.Label == "Positive";
head(data(idx,:))
```

```
ans=8×2 table
        Word          Label
    ______________    ________

    "a+"              Positive
    "abound"          Positive
    "abounds"         Positive
    "abundance"       Positive
    "abundant"        Positive
    "accessable"      Positive
    "accessible"      Positive
    "acclaim"         Positive
```

View the first few words labeled as negative.

```
idx = data.Label == "Negative";
head(data(idx,:))
```

```
ans=8×2 table
        Word           Label
    ______________    ________

    "2-faced"         Negative
    "2-faces"         Negative
    "abnormal"        Negative
    "abolish"         Negative
    "abominable"      Negative
    "abominably"      Negative
    "abominate"       Negative
    "abomination"     Negative
```

Prepare Data for Training

To train the sentiment classifier, convert the words to word vectors using the pretrained word embedding `emb`. First remove the words that do not appear in the word embedding `emb`.

```
idx = ~isVocabularyWord(emb,data.Word);
data(idx,:) = [];
```

Set aside 10% of the words at random for testing.

```
numWords = size(data,1);
cvp = cvpartition(numWords,'HoldOut',0.1);
dataTrain = data(training(cvp),:);
dataTest = data(test(cvp),:);
```

Convert the words in the training data to word vectors using `word2vec`.

```
wordsTrain = dataTrain.Word;
XTrain = word2vec(emb,wordsTrain);
YTrain = dataTrain.Label;
```

Train Sentiment Classifier

Train a support vector machine (SVM) classifier which classifies word vectors into positive and negative categories.

```
mdl = fitcsvm(XTrain,YTrain);
```

Test Classifier

Convert the words in the test data to word vectors using `word2vec`.

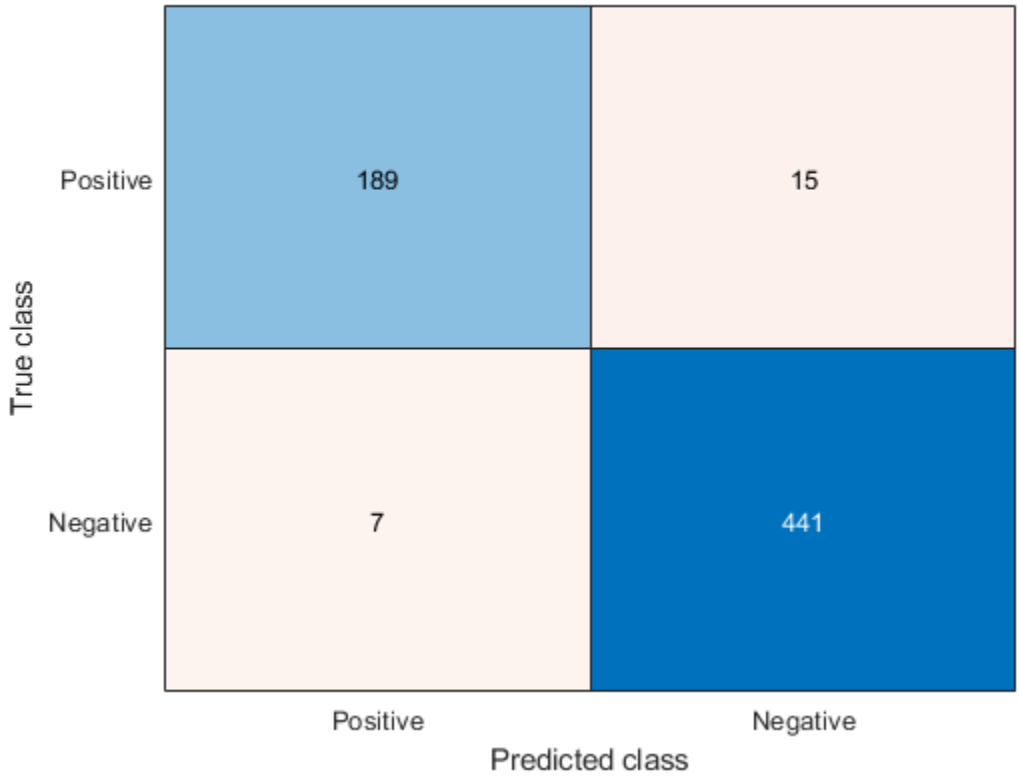
```
wordsTest = dataTest.Word;
XTest = word2vec(emb,wordsTest);
YTest = dataTest.Label;
```

Predict the sentiment labels of the test word vectors.

```
[YPred,scores] = predict(mdl,XTest);
```

Visualize the classification accuracy in a confusion matrix.

```
figure
confusionchart(YTest,YPred);
```



Visualize the classifications in word clouds. Plot the words with positive and negative sentiments in word clouds with word sizes corresponding to the prediction scores.

```
figure
subplot(1,2,1)
idx = YPred == "Positive";
wordcloud(wordsTest(idx),scores(idx,1));
title("Predicted Positive Sentiment")

subplot(1,2,2)
wordcloud(wordsTest(~idx),scores(~idx,2));
title("Predicted Negative Sentiment")
```

Calculate Sentiment of Collections of Text

To calculate the sentiment of a piece of text, for example a review, predict the sentiment score of each word in the text and take the mean sentiment score.

Load the Airbnb Summary Review data (Boston, Massachusetts, United States, 06 October, 2017) from http://insideairbnb.com/get-the-data.html. Read the data into a table and specify to read the text data as string.

```
filename = "reviews.csv";
dataReviews = readtable(filename,'TextType','string');
```

Extract the text data from the `comments` variable and view the first few reviews.

```
textData = dataReviews.comments;
textData(1:10)
```

```
ans = 10×1 string array
    "Pretty nice, quiet, cozy place to stay. Toiletries, snacks, coffee, WiFi, cable TV
    "The host was extremely welcoming and obliging. The neighborhood is quiet and charm
    "Nice and easy stay - with good accommodations especially the cable TV "
    "The host has been very accommodating and helpful. The description in the ad is acc
    "It's a great quiet stay."
    "Couldn't have been happier. The apartment was well renovated, very clean and conve
    "The apartment is very nice- as described and very convenient. The real superstar c
    "This is a brand new gorgeous place, very clean, bright and welcoming. Estee especi
    "Estee and Josh are great hosts. Very welcoming. Made us feel like we were staying
    "Estee was super sweet and so very accommodating! The apartment was nicely renovate
```

Create a function which tokenizes and preprocesses the text data so it can be used for analysis. The function `preprocessReviews`, listed at the end of the example, performs the following steps in order:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove stop words (such as "and", "of", and "the") using `removeStopWords`.

Use the preprocessing function `preprocessReviews` to prepare the text data. This step can take a few minutes to run.

```
documents = preprocessReviews(textData);
```

Remove the words from the documents that do not appear in the word embedding `emb`.

```
idx = ~isVocabularyWord(emb,documents.Vocabulary);
documents = removeWords(documents,idx);
```

To visualize how well the sentiment classifier generalizes to the reviews, classify the sentiments on the words that occur in the reviews, but not in the training data and visualize them in word clouds. Use the word clouds to manually check that the classifier behaves as expected.

```
words = documents.Vocabulary;
words(ismember(words,wordsTrain)) = [];
```

```
vec = word2vec(emb,words);
[YPred,scores] = predict(mdl,vec);

figure
subplot(1,2,1)
idx = YPred == "Positive";
wordcloud(words(idx),scores(idx,1));
title("Predicted Positive Sentiment")

subplot(1,2,2)
wordcloud(words(~idx),scores(~idx,2));
title("Predicted Negative Sentiment")
```

To calculate the sentiment of a given piece of text, compute the sentiment score for each word in the text and calculate the mean sentiment score.

For a selection of the documents, calculate the mean sentiment score. For each document, convert the words to word vectors, predict the sentiment score on the word vectors, transform the scores using the score-to-posterior transform function and then calculate the mean sentiment score.

```
idx = [7 34 331 1788 1820 1831 2185 21892 63734 76832 113276 120210];
for i = 1:numel(idx)
    words = string(documents(idx(i)));
    vec = word2vec(emb,words);
    [~,scores] = predict(mdl,vec);
    sentimentScore(i) = mean(scores(:,1));
end
```

View the predicted sentiment scores with the text data. Scores greater than 0 correspond to positive sentiment, scores less than 0 correspond to negative sentiment, and scores close to 0 correspond to neutral sentiment.

```
[sentimentScore' textData(idx)]
```

```
ans = 12×2 string array
    "0.85721"      "The apartment is very nice- as described and very convenient. The r
    "2.0453"       "Estee was the perfect Airbnb host. The apartment was comfortable, s
    "-0.37918"     "The apartment is not apropriate for 5 people. Is too little and We
    "0.94799"      "Truly a quaint place in Beacon Hill. Comfortable walking distance f
    "-0.077053"    "the neibourhood is perfect!!!!!. as it is very close to Bowdoin T S
    "0.17846"      "Although we didn't meet JJ, we felt he was very quick to respond. C
    "-0.31603"     "In the apartment it was very dirty .↵we walked in and there instinc
    "-4.0895"      "Blackmail!"
    "1.658"        "Outstanding stay.  The apartment is world-class - very, very nice.
    "1.7102"       "I had an amazing stay at Carney's. The hosts are friendly and very
    "0.67654"      "My husband and I came to Boston for our 1 year anniversary. We're s
    "-0.21651"     "My fiancÃ© and I had just gotten engaged and wanted to stay somewhe
```

Sentiment Lexicon Reading Function

This function reads the positive and negative words from the sentiment lexicon and returns a table. The table contains variables `Word` and `Label`, where `Label` contains categorical values `Positive` and `Negative` corresponding to the sentiment of each word.

```
function data = readLexicon

% Read positive words
fidPositive = fopen(fullfile('opinion-lexicon-English','positive-words.txt'));
C = textscan(fidPositive,'%s','CommentStyle',';');
wordsPositive = string(C{1});

% Read negative words
fidNegative = fopen(fullfile('opinion-lexicon-English','negative-words.txt'));
C = textscan(fidNegative,'%s','CommentStyle',';');
wordsNegative = string(C{1});
fclose all;

% Create table of labeled words
words = [wordsPositive;wordsNegative];
labels = categorical(nan(numel(words),1));
labels(1:numel(wordsPositive)) = "Positive";
labels(numel(wordsPositive)+1:end) = "Negative";

data = table(words,labels,'VariableNames',{'Word','Label'});

end
```

Preprocessing Function

The function `preprocessReviews` performs the following steps:

1. Convert the text data to lowercase using `lower`.
2. Tokenize the text using `tokenizedDocument`.
3. Erase punctuation using `erasePunctuation`.
4. Remove stop words (such as "and", "of", and "the") using `removeStopWords`.

```
function [documents] = preprocessReviews(textData)

% Convert the text data to lowercase.
cleanTextData = lower(textData);

% Tokenize the text.
documents = tokenizedDocument(cleanTextData);

% Erase punctuation.
documents = erasePunctuation(documents);
```

```
% Remove a list of stop words.
documents = removeStopWords(documents);

end
```

See Also

`bagOfWords` | `erasePunctuation` | `fastTextWordEmbedding` | `removeStopWords` | `removeWords` | `tokenizedDocument` | `word2vec` | `wordcloud`

Related Examples

- “Create Simple Text Model for Classification” on page 2-2
- “Analyze Text Data Containing Emojis” on page 2-25
- “Analyze Text Data Using Topic Models” on page 2-18
- “Analyze Text Data Using Multiword Phrases” on page 2-9
- “Classify Text Data Using Deep Learning” on page 2-43
- “Generate Text Using Deep Learning” (Deep Learning Toolbox)

Classify Text Data Using Deep Learning

This example shows how to classify text descriptions of weather reports using a deep learning long short-term memory (LSTM) network.

Text data is naturally sequential. A piece of text is a sequence of words, which might have dependencies between them. To learn and use long-term dependencies to classify sequence data, use an LSTM neural network. An LSTM network is a type of recurrent neural network (RNN) that can learn long-term dependencies between time steps of sequence data.

To input text to an LSTM network, first convert the text data into numeric sequences. You can achieve this using a word encoding which maps documents to sequences of numeric indices. For better results, also include a word embedding layer in the network. Word embeddings map words in a vocabulary to numeric vectors rather than scalar indices. These embeddings capture semantic details of the words, so that words with similar meanings have similar vectors. They also model relationships between words through vector arithmetic. For example, the relationship "*king is to queen as man is to woman*" is described by the equation *king – man + woman = queen*.

There are four steps in training and using the LSTM network in this example:

- Import and preprocess the data.
- Convert the words to numeric sequences using a word encoding.
- Create and train an LSTM network with a word embedding layer.
- Classify new text data using the trained LSTM network.

Import Data

Import the weather reports data. This data contains labeled textual descriptions of weather events. To import the text data as strings, specify the text type to be `'string'`.

```
filename = "weatherReports.csv";
data = readtable(filename,'TextType','string');
head(data)
```

```
ans=8×16 table
            Time             event_id          state              event_type          da
    _____________________    __________    _______________    ____________________    __

    22-Jul-2016 16:10:00    6.4433e+05    "MISSISSIPPI"       "Thunderstorm Wind"
```

```
    15-Jul-2016 17:15:00    6.5182e+05    "SOUTH CAROLINA"    "Heavy Rain"
    15-Jul-2016 17:25:00    6.5183e+05    "SOUTH CAROLINA"    "Thunderstorm Wind"
    16-Jul-2016 12:46:00    6.5183e+05    "NORTH CAROLINA"    "Thunderstorm Wind"
    15-Jul-2016 14:28:00    6.4332e+05    "MISSOURI"          "Hail"
    15-Jul-2016 16:31:00    6.4332e+05    "ARKANSAS"          "Thunderstorm Wind"
    15-Jul-2016 16:03:00    6.4343e+05    "TENNESSEE"         "Thunderstorm Wind"
    15-Jul-2016 17:27:00    6.4344e+05    "TENNESSEE"         "Hail"
```

Remove the rows of the table with empty reports.

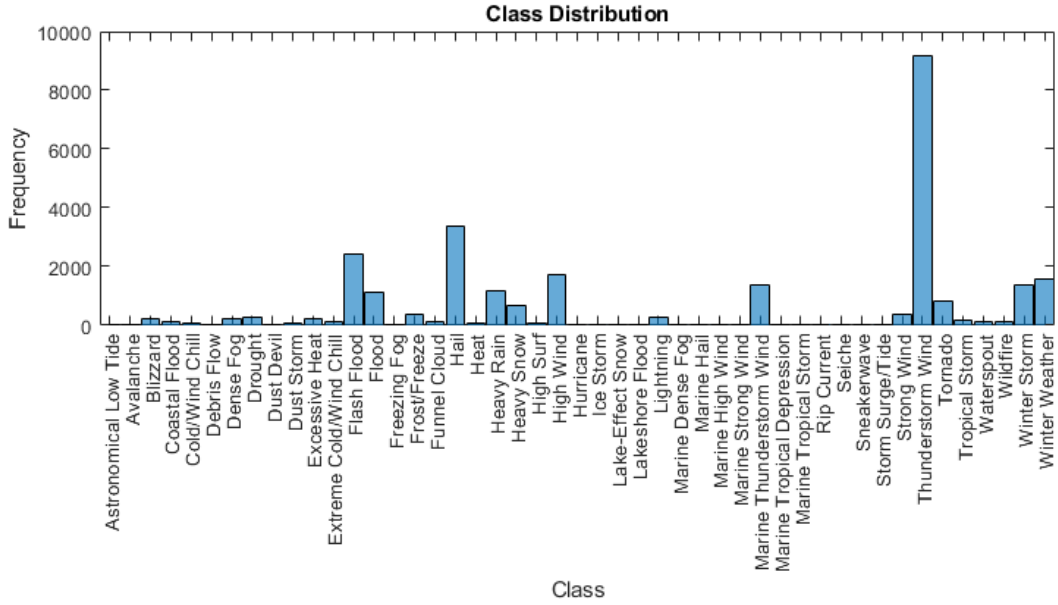
```
idxEmpty = strlength(data.event_narrative) == 0;
data(idxEmpty,:) = [];
```

The goal of this example is to classify events by the label in the `event_type` column. To divide the data into classes, convert these labels to categorical.

```
data.event_type = categorical(data.event_type);
```

View the distribution of the classes in the data using a histogram. To make the labels easier to read, increase the width of the figure.

```
f = figure;
f.Position(3) = 1.5*f.Position(3);

h = histogram(data.event_type);
xlabel("Class")
ylabel("Frequency")
title("Class Distribution")
```



The classes of the data are imbalanced, with many classes containing few observations. When the classes are imbalanced in this way, the network might converge to a less accurate model. To prevent this problem, remove any classes which appear fewer than ten times.

Get the frequency counts of the classes and the class names from the histogram.

```
classCounts = h.BinCounts;
classNames = h.Categories;
```

Find the classes containing fewer than ten observations.

```
idxLowCounts = classCounts < 10;
infrequentClasses = classNames(idxLowCounts)
```

```
infrequentClasses = 1×8 cell array
    {'Freezing Fog'}    {'Hurricane'}    {'Lakeshore Flood'}    {'Marine Dense Fog'}
```

Remove these infrequent classes from the data. Use `removecats` to remove the unused categories from the categorical data.

```
idxInfrequent = ismember(data.event_type,infrequentClasses);
data(idxInfrequent,:) = [];
data.event_type = removecats(data.event_type);
```

Now the data is sorted into classes of reasonable size. The next step is to partition it into sets for training, validation, and testing. Partition the data into a training partition and a held-out partition for validation and testing. Specify the holdout percentage to be 30%.

```
cvp = cvpartition(data.event_type,'Holdout',0.3);
dataTrain = data(training(cvp),:);
dataHeldOut = data(test(cvp),:);
```

Partition the held-out set again to get a validation set. Specify the holdout percentage to be 50%. This results in a partitioning of 70% training observations, 15% validation observations, and 15% test observations.

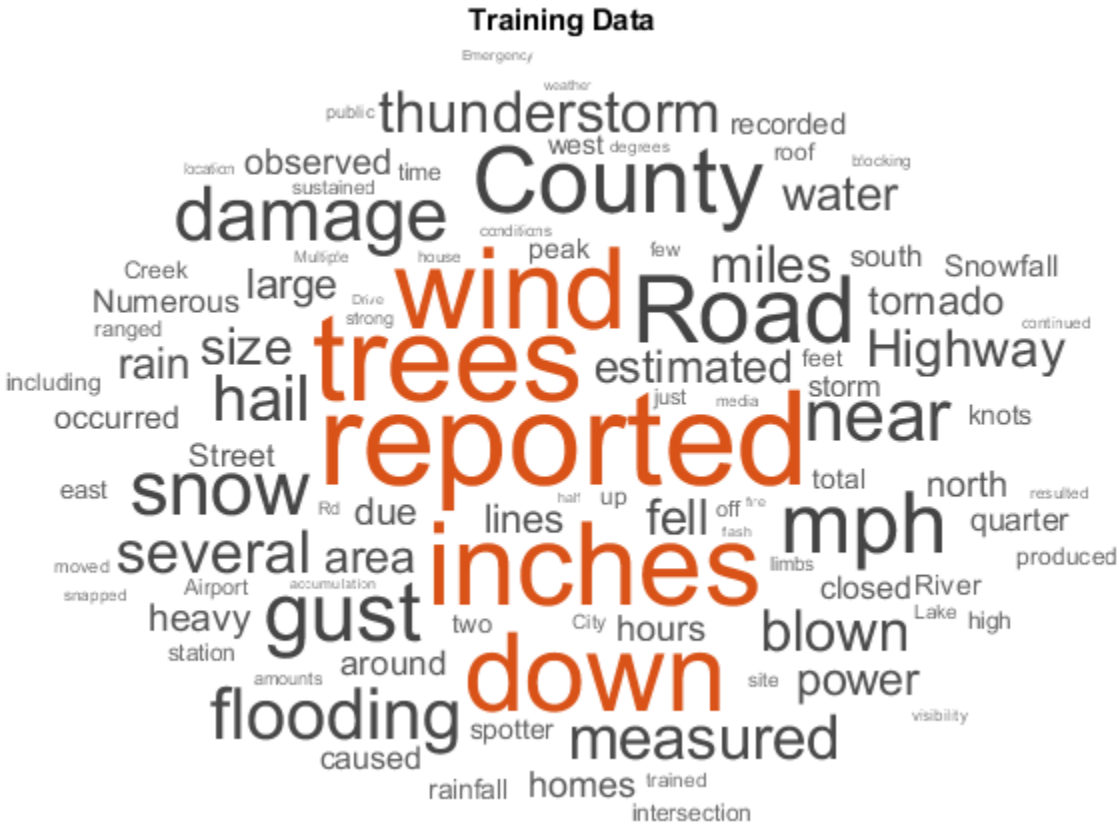
```
cvp = cvpartition(dataHeldOut.event_type,'HoldOut',0.5);
dataValidation = dataHeldOut(training(cvp),:);
dataTest = dataHeldOut(test(cvp),:);
```

Extract the text data and labels from the partitioned tables.

```
textDataTrain = dataTrain.event_narrative;
textDataValidation = dataValidation.event_narrative;
textDataTest = dataTest.event_narrative;
YTrain = dataTrain.event_type;
YValidation = dataValidation.event_type;
YTest = dataTest.event_type;
```

To check that you have imported the data correctly, visualize the training text data using a word cloud.

```
figure
wordcloud(textDataTrain);
title("Training Data")
```



Preprocess Text Data

Preprocess the training data. Convert the text to lowercase, tokenize it, and then erase the punctuation. Do not stem or remove words, as these steps can lead to a worse word embedding fit.

```
textDataTrain = lower(textDataTrain);
documentsTrain = tokenizedDocument(textDataTrain);
documentsTrain = erasePunctuation(documentsTrain);

textDataValidation = lower(textDataValidation);
documentsValidation = tokenizedDocument(textDataValidation);
documentsValidation = erasePunctuation(documentsValidation);
```

View the first few preprocessed training documents.

```
documentsTrain(1:5)
```

```
ans =
  5×1 tokenizedDocument:

    (1,1)   7 tokens: large tree down between plantersville and nettleton
    (2,1)  37 tokens: one to two feet of deep standing water developed on a stre…
    (3,1)  13 tokens: nws columbia relayed a report of trees blown down along to…
    (4,1)  13 tokens: media reported two trees blown down along i40 in the old f…
    (5,1)  14 tokens: a few tree limbs greater than 6 inches down on hwy 18 in r…
```

Convert Document to Sequences

To input the documents into an LSTM network, use a word encoding to convert the documents into sequences of numeric indices.

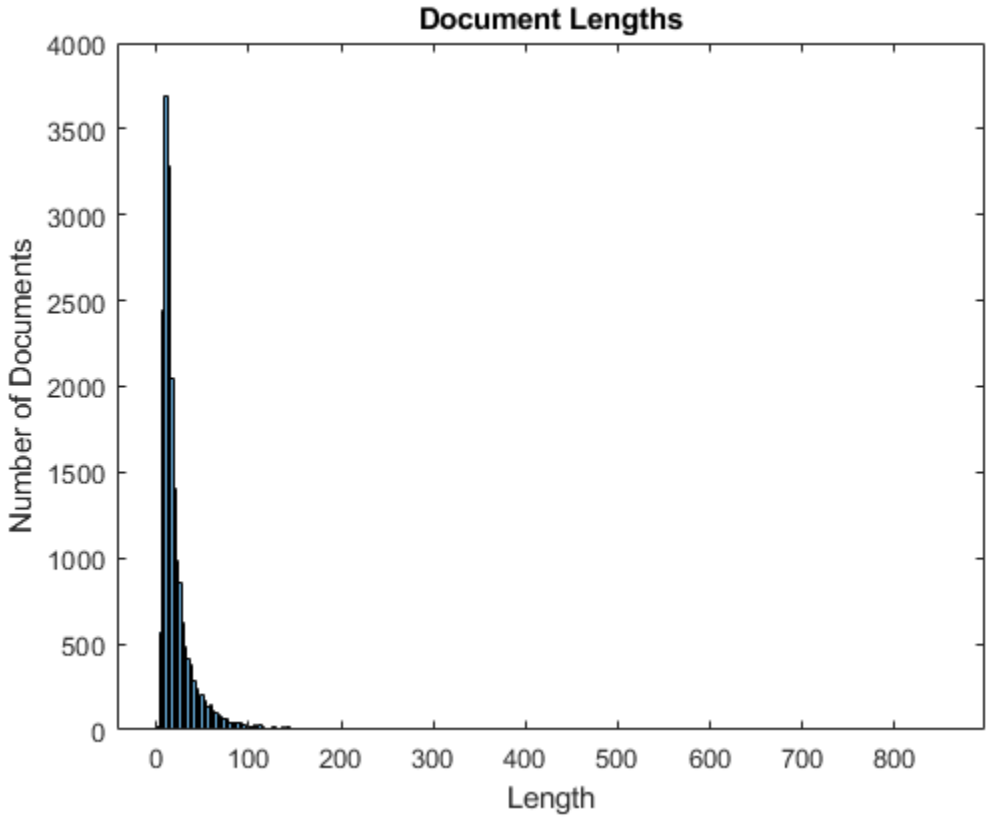
To create a word encoding, use the `wordEncoding` function.

```
enc = wordEncoding(documentsTrain);
```

The next conversion step is to pad and truncate documents so they are all the same length. The `trainingOptions` function provides options to pad and truncate input sequences automatically. However, these options are not well suited for sequences of word vectors. Instead, pad and truncate the sequences manually. If you *left-pad* and truncate the sequences of word vectors, then the training might improve.

To pad and truncate the documents, first choose a target length, and then truncate documents that are longer than it and left-pad documents that are shorter than it. For best results, the target length should be short without discarding large amounts of data. To find a suitable target length, view a histogram of the training document lengths.

```
documentLengths = doclength(documentsTrain);
figure
histogram(documentLengths)
title("Document Lengths")
xlabel("Length")
ylabel("Number of Documents")
```



Most of the training documents have fewer than 75 tokens. Use this as your target length for truncation and padding.

Convert the documents to sequences of numeric indices using `doc2sequence`. To truncate or left-pad the sequences to have length 75, set the `'Length'` option to 75.

```
XTrain = doc2sequence(enc,documentsTrain,'Length',75);
XTrain(1:5)
```

```
ans = 5×1 cell array
    {1×75 double}
    {1×75 double}
    {1×75 double}
    {1×75 double}
```

```
    {1×75 double}
```

Convert the validation documents to sequences using the same options.

```
XValidation = doc2sequence(enc,documentsValidation,'Length',75);
```

Create and Train LSTM Network

Define the LSTM network architecture. To input sequence data into the network, include a sequence input layer and set the input size to 1. Next, include a word embedding layer of dimension 100 and the same number of words as the word encoding. Next, include an LSTM layer and set the number of hidden units to 180. To use the LSTM layer for a sequence-to-label classification problem, set the output mode to `'last'`. Finally, add a fully connected layer with the same size as the number of classes, a softmax layer, and a classification layer.

```
inputSize = 1;
embeddingDimension = 100;
numHiddenUnits = enc.NumWords;
hiddenSize = 180;
numClasses = numel(categories(YTrain));

layers = [ ...
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(embeddingDimension,numHiddenUnits)
    lstmLayer(hiddenSize,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer]
```

```
layers =
  6x1 Layer array with layers:

     1   ''   Sequence Input          Sequence input with 1 dimensions
     2   ''   Word Embedding Layer    Word embedding layer with 100 dimensions and 169
     3   ''   LSTM                   LSTM with 180 hidden units
     4   ''   Fully Connected         39 fully connected layer
     5   ''   Softmax                softmax
     6   ''   Classification Output   crossentropyex
```

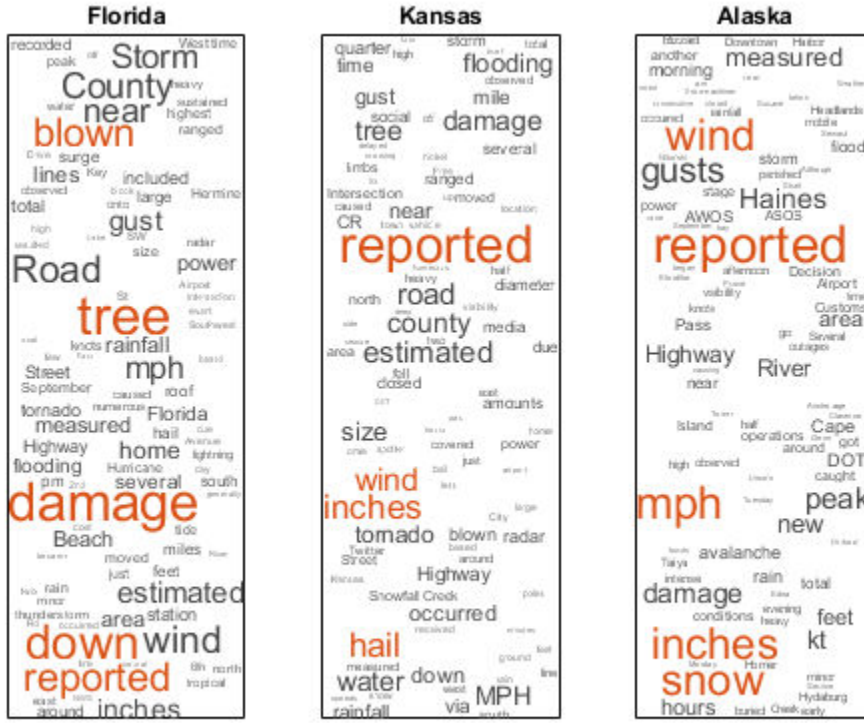
Specify the training options. Set the solver to `'adam'`, train for 10 epochs, and set the gradient threshold to 1. Set the initial learn rate to 0.01. To monitor the training progress, set the `'Plots'` option to `'training-progress'`. Specify the validation data using the `'ValidationData'` option. To suppress verbose output, set `'Verbose'` to `false`.

```
wordcloud(textData(idx),'Color','magenta');
title("Thunderstorm Wind")
```

Compare the words in the reports from the states Florida, Kansas, and Alaska. Create word clouds of the reports for each of these states in rectangles and draw a border around each word cloud.

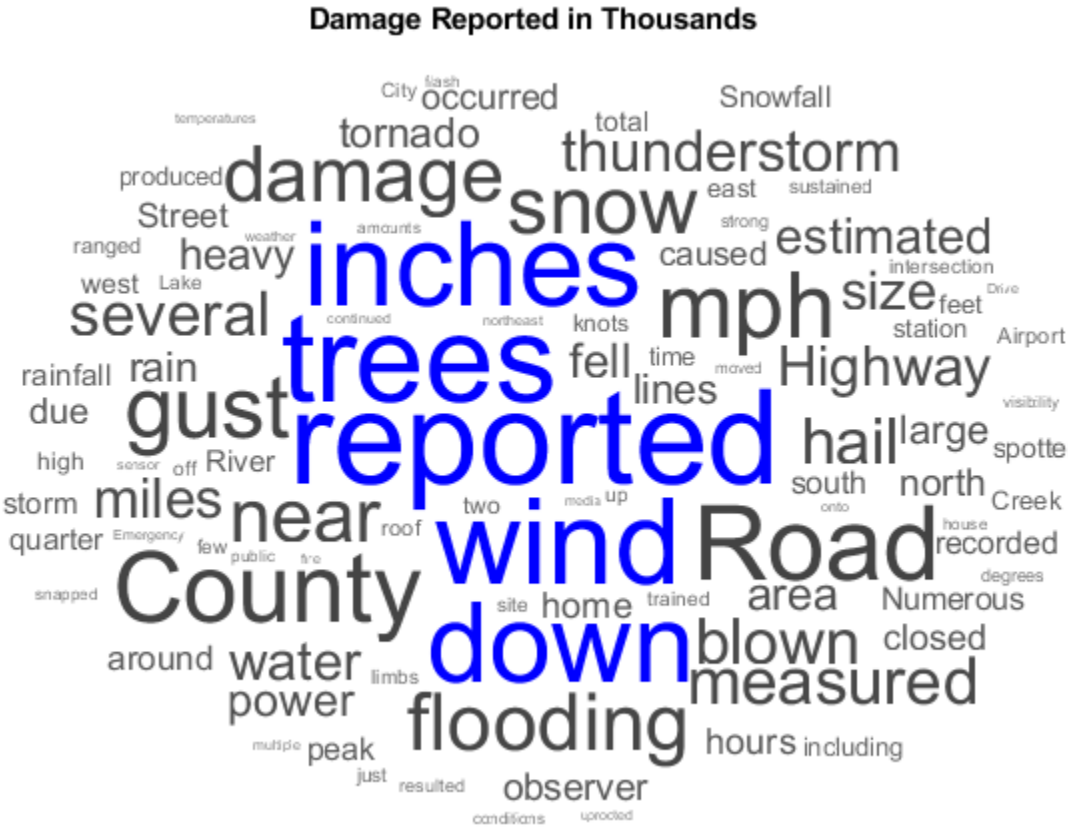
```
figure
state = T.state;

subplot(1,3,1)
idx = state == "FLORIDA";
wordcloud(textData(idx),'Shape','rectangle','Box','on');
title("Florida")
```

```
subplot(1,3,2)
idx = state == "KANSAS";
wordcloud(textData(idx),'Shape','rectangle','Box','on');
title("Kansas")

subplot(1,3,3)
idx = state == "ALASKA";
wordcloud(textData(idx),'Shape','rectangle','Box','on');
title("Alaska")
```



Compare the words in the reports with property damage reported in thousands of dollars to the reports with damage reported in millions of dollars. Create word clouds of the reports for each of these amounts with highlight color blue and red respectively.

```
cost = T.damage_property;
idx = endsWith(cost,"K");
figure
wordcloud(textData(idx),'HighlightColor','blue');
title("Damage Reported in Thousands")
```




```
idx = endsWith(cost,"M");
figure
wordcloud(textData(idx),'HighlightColor','red');
title("Damage Reported in Millions")
```

See Also

`bagOfWords` | `tokenizedDocument` | `wordcloud`

Related Examples

- “Prepare Text Data for Analysis” on page 1-11
- “Analyze Text Data Using Topic Models” on page 2-18
- “Classify Text Data Using Deep Learning” on page 2-43
- “Visualize Word Embeddings Using Text Scatter Plots” on page 3-8

Visualize Word Embeddings Using Text Scatter Plots

This example shows how to visualize word embeddings using 2-D and 3-D t-SNE and text scatter plots.

Word embeddings, 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 like "Italy is to France as Rome is to Paris". In vector form, this relationship is $Italy - Rome + Paris = France$.

To reproduce the results in this example, set `rng` to `'default'`.

```
rng('default')
```

Load Pretrained Word Embedding

Load a pretrained word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox™ Model *for fastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, then the function provides a download link.

```
emb = fastTextWordEmbedding
```

```
emb =
  wordEmbedding with properties:

     Dimension: 300
    Vocabulary: [1×1000000 string]
```

Explore the word embedding using `word2vec` and `vec2word`. Convert the words *Italy*, *Rome*, and *Paris* to vectors using `word2vec`.

```
italy = word2vec(emb,"Italy");
rome = word2vec(emb,"Rome");
paris = word2vec(emb,"Paris");
```

Compute the vector given by `italy - rome + paris`. This vector encapsulates the semantic meaning of the word *Italy*, without the semantics of the word *Rome*, and also includes the semantics of the word *Paris*.

```
vec = italy - rome + paris
```

```
vec = 1×300 single row vector

    0.1606   -0.0690    0.1183   -0.0349    0.0672    0.0907   -0.1820   -0.0080    0.(
```

Find the closest words in the embedding to `vec` using `vec2word`.

```
word = vec2word(emb,vec)
```

```
word =
"France"
```

Create 2-D Text Scatter Plot

Visualize the word embedding by creating a 2-D text scatter plot using `tsne` and `textscatter`.

Convert the first 500 words to vectors using `word2vec`. `V` is a matrix of word vectors of length 300.

```
words = emb.Vocabulary(1:5000);
V = word2vec(emb,words);
size(V)
```

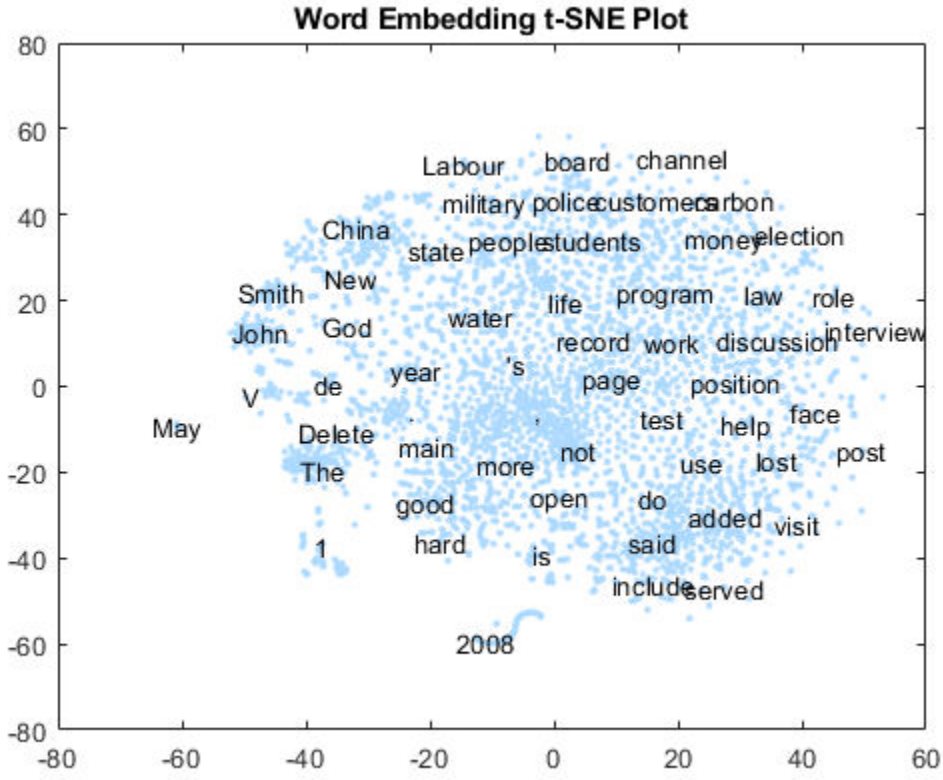
```
ans = 1×2

        5000         300
```

Embed the word vectors in two-dimensional space using `tsne`. This function may take a few minutes to run. If you want to display the convergence information, then set the `'Verbose'` name-value pair to 1.

```
XY = tsne(V);
```

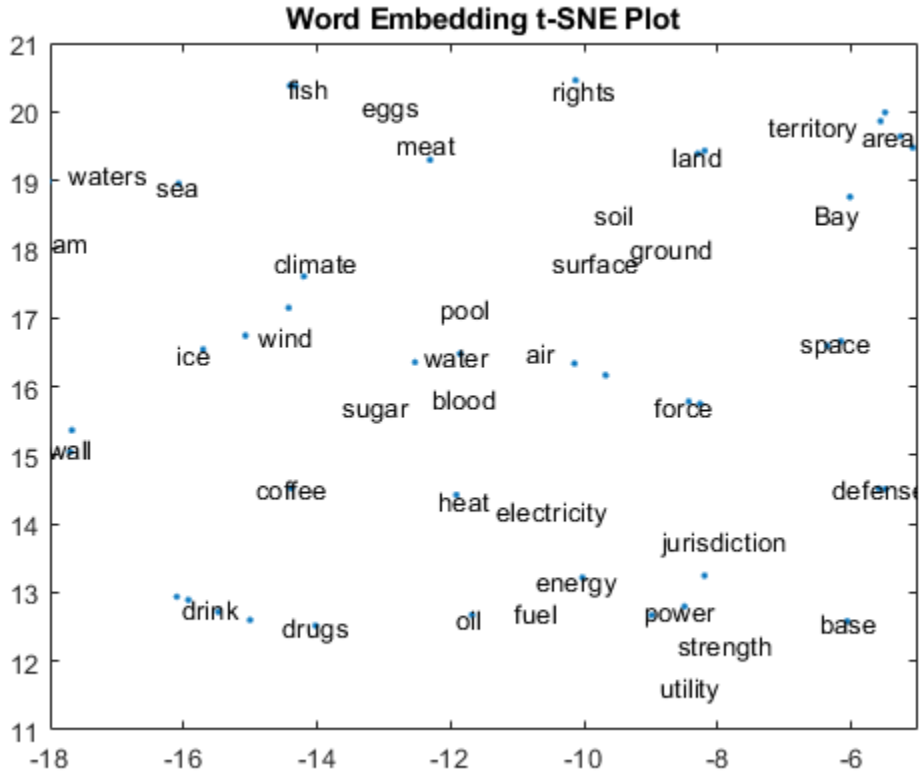
Plot the words at the coordinates specified by `XY` in a 2-D text scatter plot. For readability, `textscatter`, by default, does not display all of the input words and displays markers instead.

```
figure
textscatter(XY,words)
title("Word Embedding t-SNE Plot")
```



Zoom in on a section of the plot.

```
xlim([-18 -5])
ylim([11 21])
```



Create 3-D Text Scatter Plot

Visualize the word embedding by creating a 3-D text scatter plot using `tsne` and `textscatter`.

Convert the first 5000 words to vectors using `word2vec`. `V` is a matrix of word vectors of length 300.

```
words = emb.Vocabulary(1:5000);
V = word2vec(emb,words);
size(V)
```

```
ans = 1×2
```

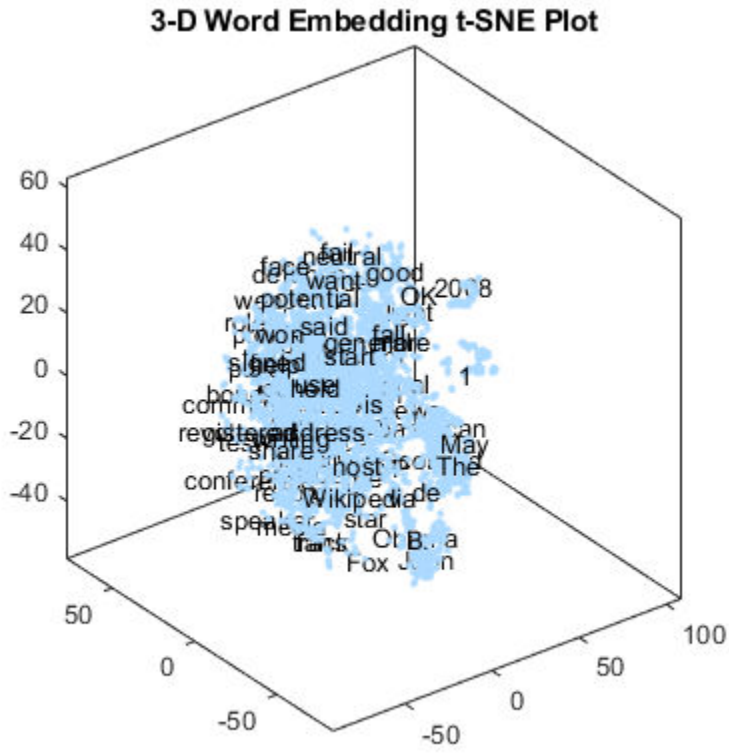
```
        5000         300
```

Embed the word vectors in a three-dimensional space using `tsne` by specifying the number of dimensions to be three. This function may take a few minutes to run. If you want to display the convergence information, then you can set the `'Verbose'` name-value pair to 1.

```
XYZ = tsne(V,'NumDimensions',3);
```

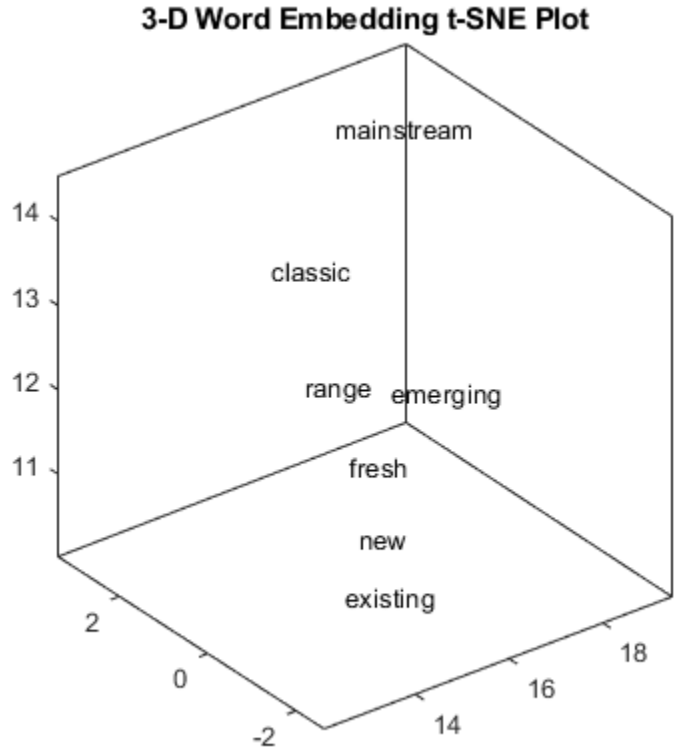
Plot the words at the coordinates specified by XYZ in a 3-D text scatter plot.

```
figure
ts = textscatter3(XYZ,words);
title("3-D Word Embedding t-SNE Plot")
```



Zoom in on a section of the plot.

```
xlim([12.04 19.48])
ylim([-2.66 3.40])
zlim([10.03 14.53])
```



Perform Cluster Analysis

Convert the first 5000 words to vectors using `word2vec`. `V` is a matrix of word vectors of length 300.

```
words = emb.Vocabulary(1:5000);
V = word2vec(emb,words);
size(V)
```

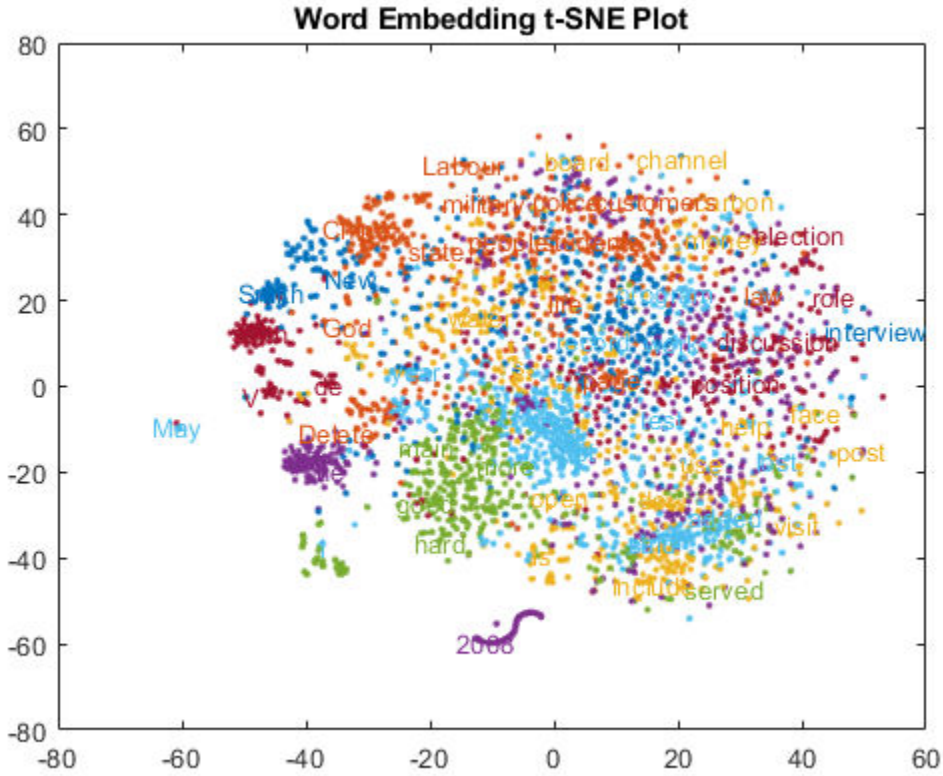
```
ans = 1×2

        5000         300
```

Discover 25 clusters using `kmeans`.

```
cidx = kmeans(V,25,'dist','sqeuclidean');
```

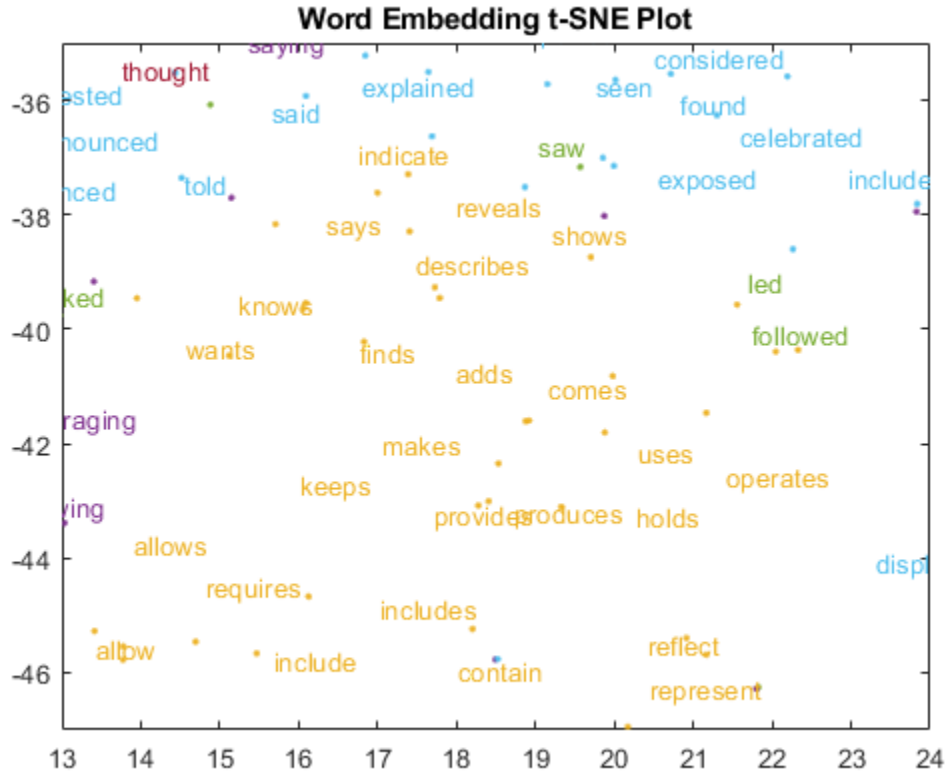
Visualize the clusters in a text scatter plot using the 2-D t-SNE data coordinates calculated earlier.

```
figure
textscatter(XY,words,'ColorData',categorical(cidx));
title("Word Embedding t-SNE Plot")
```



Zoom in on a section of the plot.

```
xlim([13 24])
ylim([-47 -35])
```



See Also

`readWordEmbedding` | `textscatter` | `textscatter3` | `tokenizedDocument` | `vec2word` | `word2vec` | `wordEmbedding`

Related Examples

- “Extract Text Data from Files” on page 1-2
- “Prepare Text Data for Analysis” on page 1-11
- “Visualize Text Data Using Word Clouds” on page 3-2
- “Classify Text Data Using Deep Learning” on page 2-43