Text Analytics Toolbox™

Examples

MATLAB®

R2018a

MathWorks®

How to Contact MathWorks

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

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

Revision History

September 2017	Online Only	New for Version 1.0
March 2018	Online Only	Revised for Version 1.1 (Release 2018a)

Contents

1 Text Analytics Toolbox Examples

1

Text Analytics Toolbox Examples

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 Files

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 Documents

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 `strrep` function.

```
sonnet2 = strrep(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 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 Code

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

```
sonnet4 = 
    "
```

```
     Unthrifty loveliness, why dost thou spend Upon thy self thy beauty’s legacy? Natu
     Then, beauteous niggard, why dost thou abuse The bounteous largess given thee to
     Then how when nature calls thee to be gone, What acceptable audit canst thou leav
     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 f
```

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 filename using the wildcard "*" to find all filenames 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` | `readPDFFormData`

Related Examples

- “Prepare Text Data for Analysis” on page 1-16
- “Create Simple Text Model for Classification” on page 1-9
- “Visualize Text Data Using Word Clouds” on page 1-26
- “Analyze Text Data Using Topic Models” on page 1-32
- “Analyze Text Data Using Multiword Phrases” on page 1-51
- “Classify Text Data Using Deep Learning” on page 1-68

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)
```

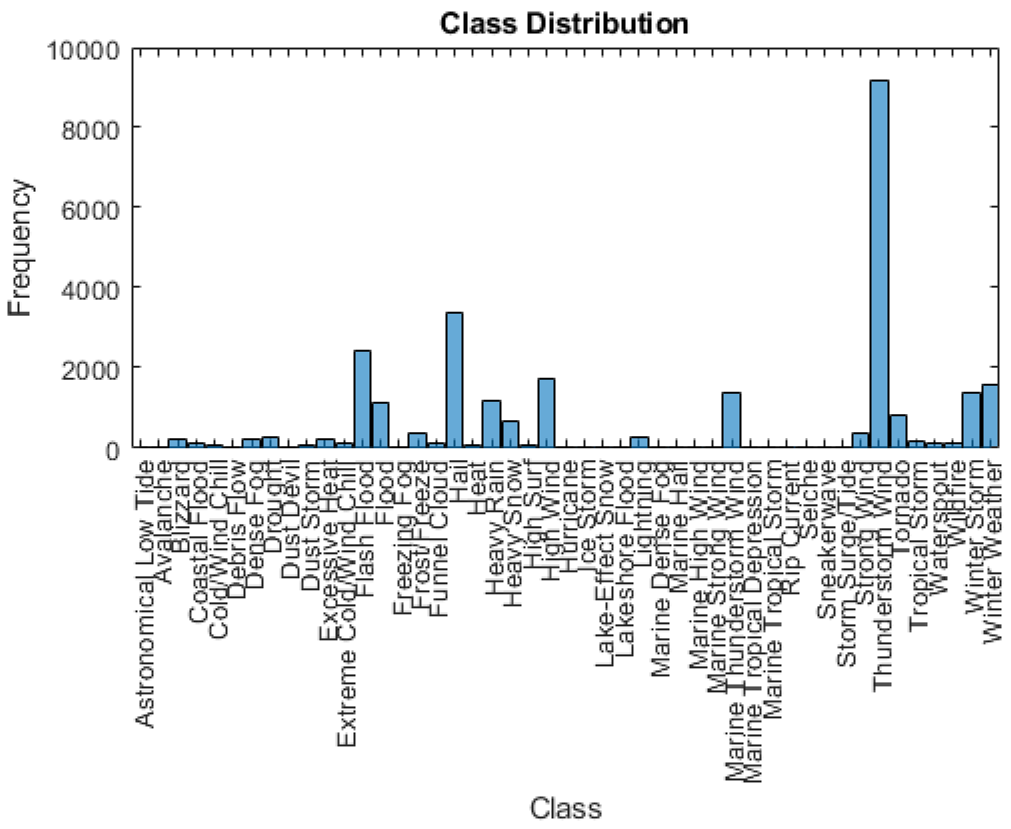
```
ans=8×16 table
            Time             event_id          state              event_type           da
    ____________________    __________    ________________    ___________________    __

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

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

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

ans = 
  5x1 tokenizedDocument:

(1,1)  5 tokens: larg tree down plantersvil nettleton
(2,1) 18 tokens: two feet deep stand water develop street winthrop univers campu inch r
(3,1)  9 tokens: nw columbia relai report tree blown down tom hall
(4,1) 10 tokens: media report two tree blown down i40 old fort area
(5,1)  8 tokens: few tree limb greater inch down hwy roseland
```

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

```
bag = bagOfWords(documents)

bag = 
  bagOfWords with properties:

          Counts: [25316x16906 double]
      Vocabulary: [1x16906 string]
        NumWords: 16906
    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: [25315x6247 double]
      Vocabulary: [1x6247 string]
        NumWords: 6247
    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: [1x39 categorical]
    ScoreTransform: 'none'
    BinaryLearners: {741x1 cell}
      CodingMatrix: [39x741 double]


  Properties, Methods
```

For a better fit, you can try specifying different parameters of the linear learners. For example, if you specify `'Learners'` to be `templateLinear('Solver','lbfgs')`, then you might experience improved accuracy at the cost of a slower fit. 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 = 0.8829
```

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 = 3x1 categorical array
     Thunderstorm Wind
     Thunderstorm Wind
     Flash Flood
```

Example Preprocessing Function

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

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

```
function documents = preprocessWeatherNarratives(textData)
% Erase punctuation.
cleanTextData = erasePunctuation(textData);

% Convert the text data to lowercase.
```

```
cleanTextData = lower(cleanTextData);

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

% Remove a list of stop words.
documents = removeWords(documents,stopWords);

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

% Normalize the words using the Porter stemmer.
documents = normalizeWords(documents);

end
```

See Also

`bagOfWords` | `normalizeWords` | `tokenizedDocument` | `wordcloud`

Related Examples

- "Prepare Text Data for Analysis" on page 1-16
- "Visualize Text Data Using Word Clouds" on page 1-26
- "Analyze Text Data Using Topic Models" on page 1-32
- "Analyze Text Data Using Multiword Phrases" on page 1-51
- "Classify Text Data Using Deep Learning" on page 1-68

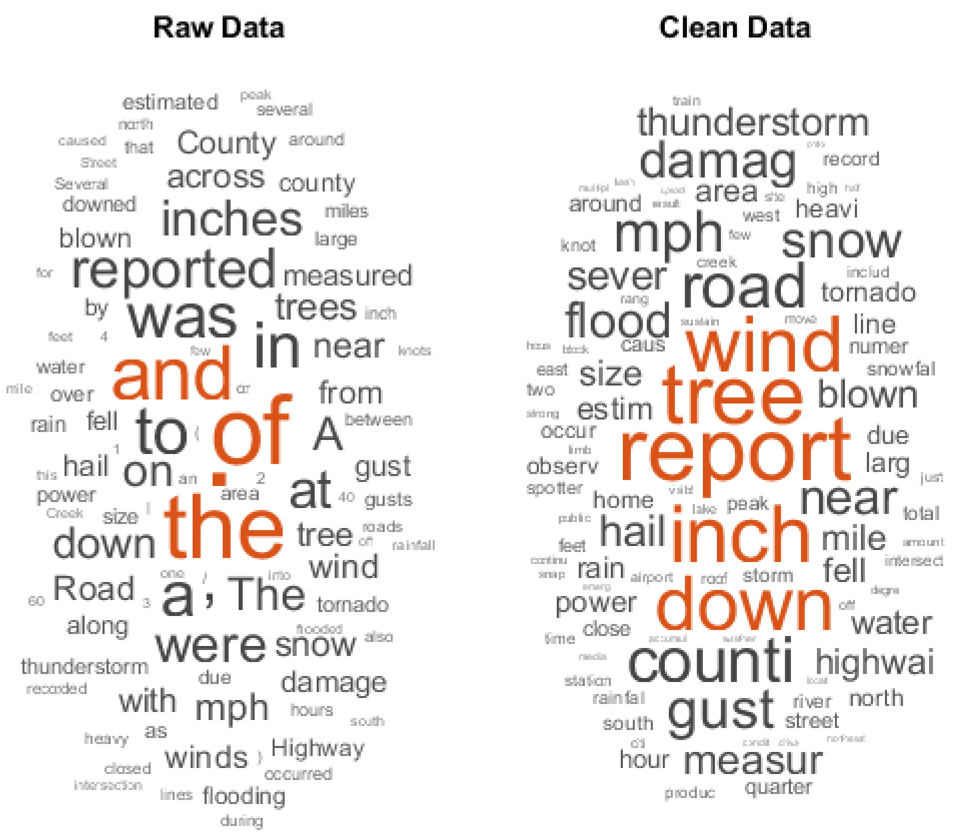
Prepare Text Data for Analysis

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 = 10x1 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 String Data for Tokenizing

Erase the punctuation from the text data.

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

```
ans = 10x1 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 I40 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 intersect
    "Quarter size hail near Rosemark"
    "Tin roof ripped off house on Old Memphis Road near Billings Drive Several large tr
    "Powerlines down at Walnut Grove and Cherry Lane roads"
```

Convert the text data to lowercase.

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

```
ans = 10x1 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 i40 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 intersect
    "quarter size hail near rosemark"
    "tin roof ripped off house on old memphis road near billings drive several large tr
    "powerlines down at walnut grove and cherry lane roads"
```

Create Tokenized Documents

Create an array of tokenized documents.

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

```
ans = 
  10x1 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 street on the w
 (3,1) 13 tokens: nws columbia relayed a report of trees blown down along tom hall st
 (4,1) 13 tokens: media reported two trees blown down along i40 in the old fort area
 (5,1)  0 tokens:
 (6,1) 14 tokens: a few tree limbs greater than 6 inches down on hwy 18 in roseland
 (7,1) 18 tokens: awning blown off a building on lamar avenue multiple trees down near
 (8,1)  5 tokens: quarter size hail near rosemark
 (9,1) 19 tokens: tin roof ripped off house on old memphis road near billings drive sev
(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 `stopWords` and `removeWords` functions.

```
cleanDocuments = removeWords(cleanDocuments,stopWords);
cleanDocuments(1:10)
```

```
ans = 
  10x1 tokenizedDocument:

 (1,1)  5 tokens: large tree down plantersville nettleton
 (2,1) 18 tokens: two feet deep standing water developed street winthrop university cam
 (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 near inter
 (8,1)  5 tokens: quarter size hail near rosemark
 (9,1) 16 tokens: tin roof ripped off house old memphis road near billings drive severa
(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 = 
  10x1 tokenizedDocument:

 (1,1)  5 tokens: large tree down plantersville nettleton
 (2,1) 18 tokens: two feet deep standing water developed street winthrop university cam
 (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 near inter
 (8,1)  5 tokens: quarter size hail near rosemark
 (9,1) 16 tokens: tin roof ripped off house old memphis road near billings drive severa
(10,1)  7 tokens: powerlines down walnut grove cherry lane roads
```

Normalize the words using the Porter stemmer.

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

```
ans = 
  10x1 tokenizedDocument:

 (1,1)  5 tokens: larg tree down plantersvil nettleton
 (2,1) 18 tokens: two feet deep stand water develop street winthrop univers campu inch
 (3,1)  9 tokens: nw columbia relai report tree blown down tom hall
 (4,1) 10 tokens: media report two tree blown down i40 old fort area
 (5,1)  0 tokens:
 (6,1)  8 tokens: few tree limb greater inch down hwy roseland
 (7,1) 13 tokens: awn blown off build lamar avenu multipl tree down near intersect wind
 (8,1)  5 tokens: quarter size hail near rosemark
 (9,1) 16 tokens: tin roof rip off hous old memphi road near bill drive sever larg tree
(10,1)  7 tokens: powerlin down walnut grove cherri lane road
```

Create Bag-of-Words Model

Create a bag-of-words model.

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

```
cleanBag = 
  bagOfWords with properties:

          Counts: [36176x17816 double]
      Vocabulary: [1x17816 string]
        NumWords: 17816
    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: [36176x6651 double]
      Vocabulary: [1x6651 string]
        NumWords: 6651
    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: [28137x6651 double]
      Vocabulary: [1x6651 string]
        NumWords: 6651
```

```
    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. Erase punctuation using `erasePunctuation`.
2. Convert the text data to lowercase using `lower`.
3. Tokenize the text using `tokenizedDocument`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeWords` and `stopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Normalize the words using the Porter stemmer 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 outsid appl 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: [36176x22720 double]
      Vocabulary: [1x22720 string]
        NumWords: 22720
    NumDocuments: 36176
```

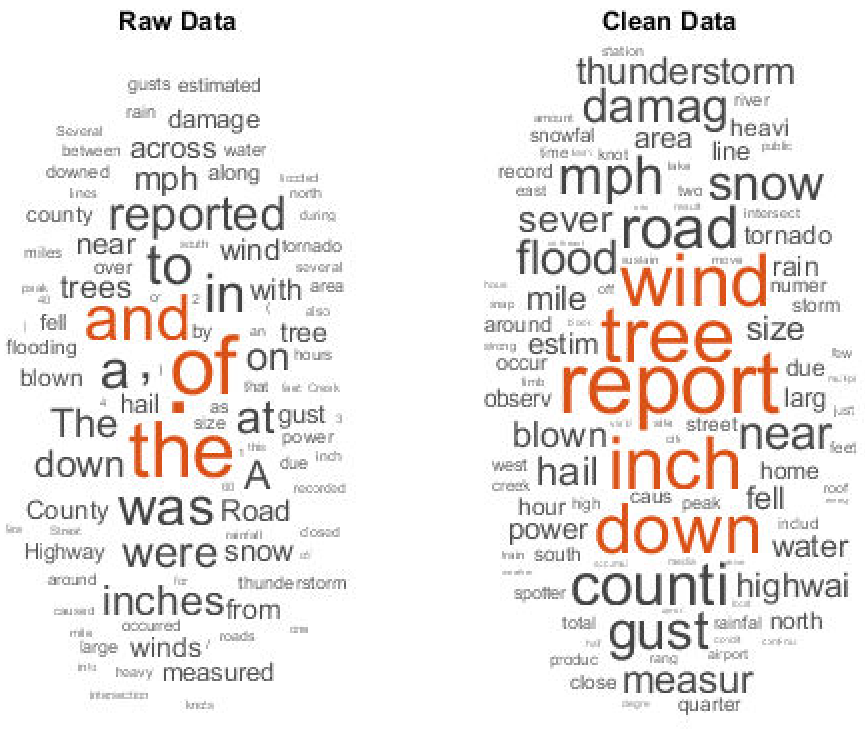
Calculate the reduction in data.

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

```
reduction = 0.7073
```

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

7 Normalize the words using the Porter stemmer using `normalizeWords`.

```
function [documents] = preprocessWeatherNarratives(textData)
% Erase punctuation.
cleanTextData = erasePunctuation(textData);

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

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

% Remove a list of stop words.
documents = removeWords(documents,stopWords);

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

% Normalize the words using the Porter stemmer.
documents = normalizeWords(documents);
end
```

See Also

`bagOfWords` | `normalizeWords` | `tokenizedDocument` | `wordcloud`

Related Examples

- “Extract Text Data from Files” on page 1-2
- “Create Simple Text Model for Classification” on page 1-9
- “Visualize Text Data Using Word Clouds” on page 1-26
- “Analyze Text Data Using Topic Models” on page 1-32
- “Analyze Text Data Using Multiword Phrases” on page 1-51
- “Classify Text Data Using Deep Learning” on page 1-68

Visualize Text Data Using Word Clouds

This example shows how to visualize text data using word clouds.

Text Analytics Toolbox extends the functionality of the `wordcloud` (MATLAB) function. It adds support for creating word clouds directly from string arrays, and creating word clouds from bag-of-words models and LDA topics.

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

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

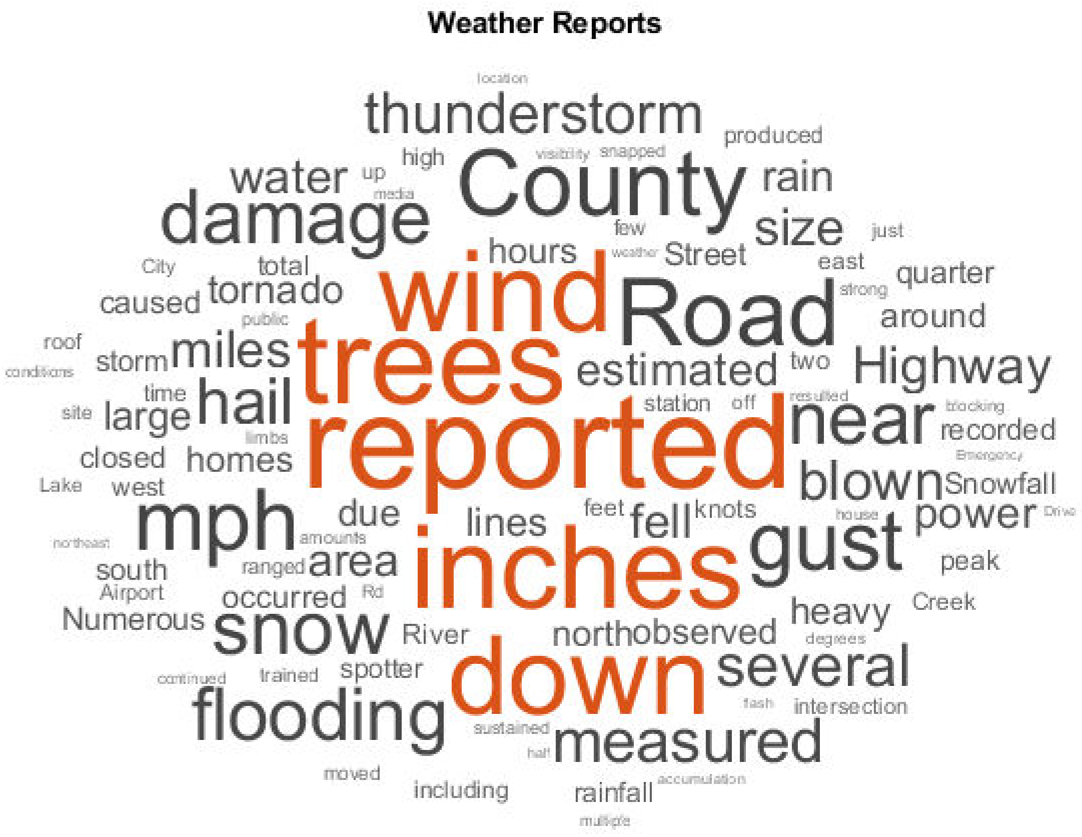
Extract the text data from the `event_narrative` column.

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

```
ans = 10x1 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 a word cloud from all the weather reports.

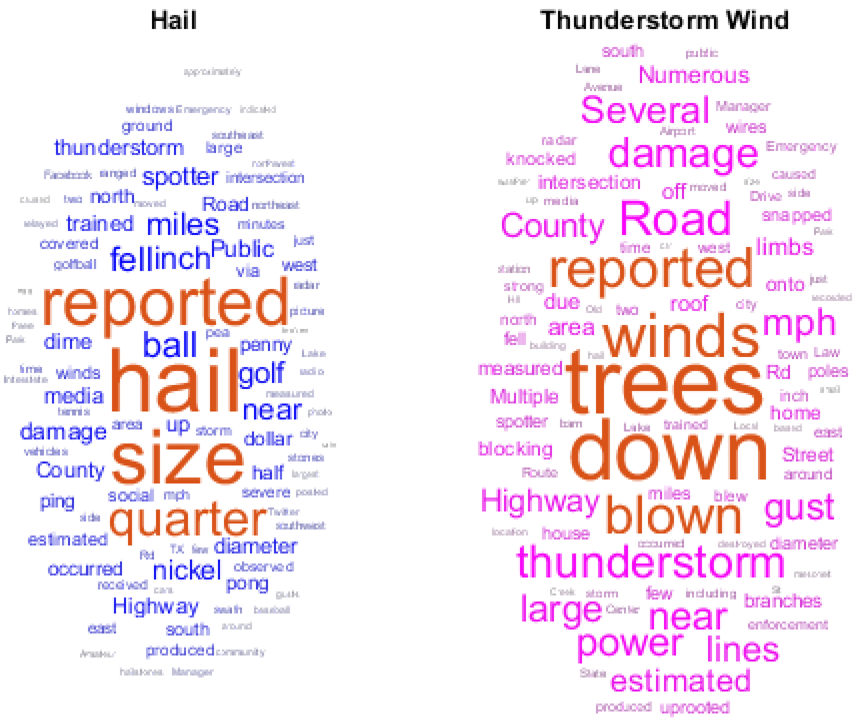
```
figure
wordcloud(textData);
title("Weather Reports")
```



Compare the words in the reports with labels `"Hail"` and `"Thunderstorm Wind"`. Create word clouds of the reports for each of these labels. Specify the word colors to be blue and magenta for each word cloud respectively.

```
figure
labels = T.event_type;

subplot(1,2,1)
idx = labels == "Hail";
wordcloud(textData(idx),'Color','blue');
title("Hail")

subplot(1,2,2)
idx = labels == "Thunderstorm Wind";
```

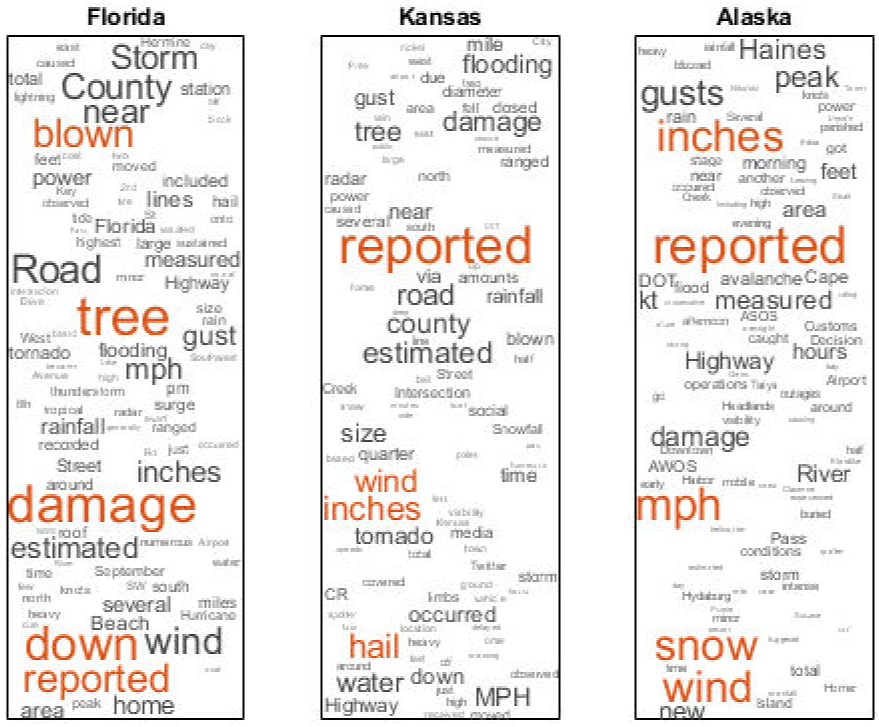
```
wordcloud(textData(idx),'Color','magenta');
title("Thunderstorm Wind")
```



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.

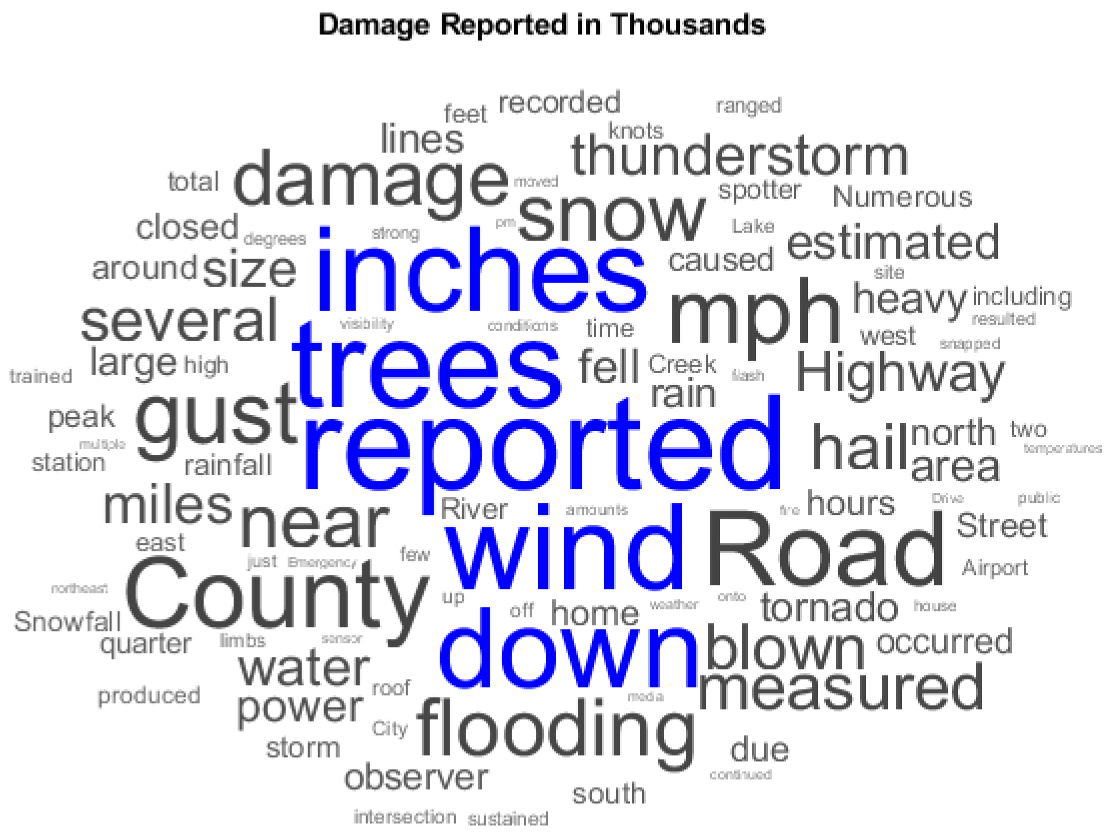
```
figure
state = T.state;

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")
```

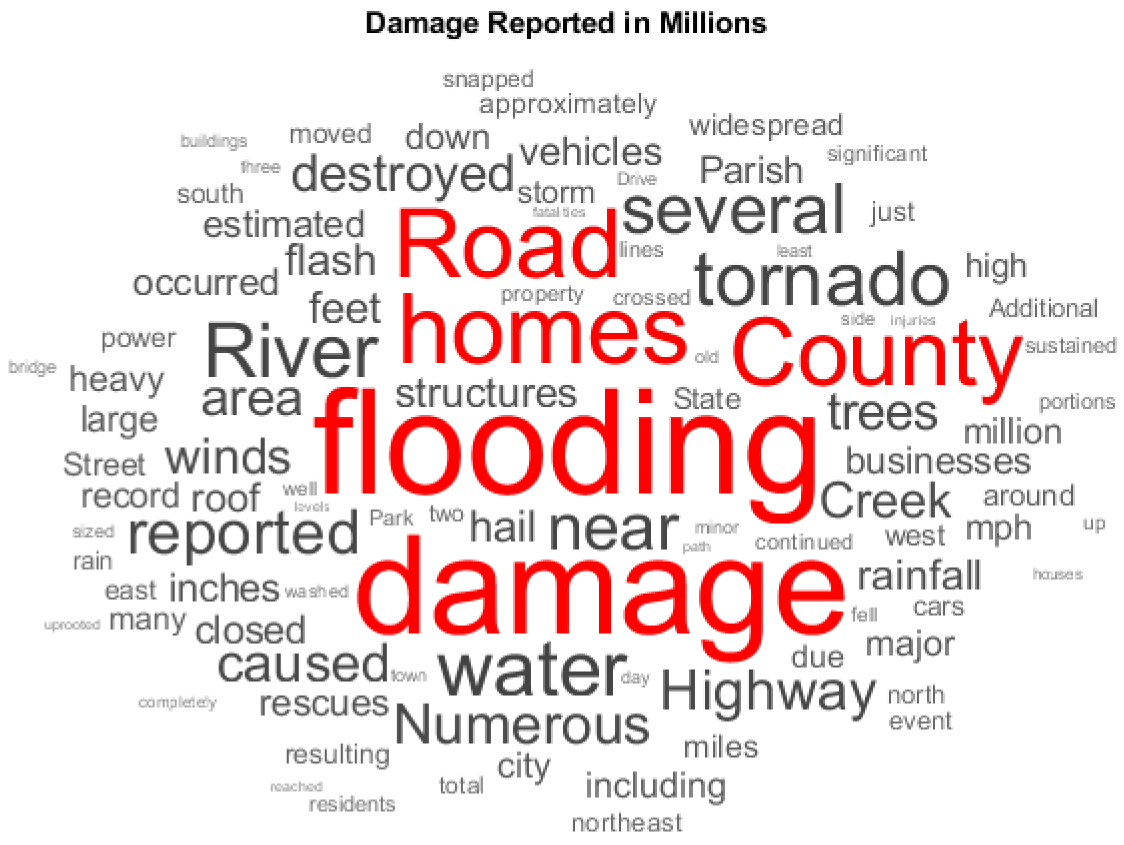


Compare the words in the reports with property damage reported in thousands of dollars to the reports with damage reported in millions of dollars. Create word clouds of the reports for each of these amounts with highlight color blue and red respectively.

```
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

`wordcloud`

Related Examples

- "Prepare Text Data for Analysis" on page 1-16
- "Analyze Text Data Using Topic Models" on page 1-32
- "Classify Text Data Using Deep Learning" on page 1-68
- "Visualize Word Embeddings Using Text Scatter Plots" on page 1-60

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 = 10x1 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. Erase punctuation using `erasePunctuation`.
2. Convert the text data to lowercase using `lower`.
3. Tokenize the text using `tokenizedDocument`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeWords` and `stopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Normalize the words using the Porter stemmer using `normalizeWords`.

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

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

```
ans = 
  5x1 tokenizedDocument:

(1,1)  5 tokens: larg tree down plantersvil nettleton
(2,1) 18 tokens: two feet deep stand water develop street winthrop univers campu inch r
(3,1)  9 tokens: nw columbia relai report tree blown down tom hall
(4,1) 10 tokens: media report two tree blown down i40 old fort area
(5,1)  0 tokens:
```

Fit LDA Model

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

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

```
bag = 
  bagOfWords with properties:

          Counts: [36176x17816 double]
      Vocabulary: [1x17816 string]
        NumWords: 17816
    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: [28137x6651 double]
      Vocabulary: [1x6651 string]
        NumWords: 6651
    NumDocuments: 28137
```

Fit an LDA model with 60 topics. For an example showing how to choose the number of topics, see “Choose Number of Topics for LDA Model” on page 1-41.

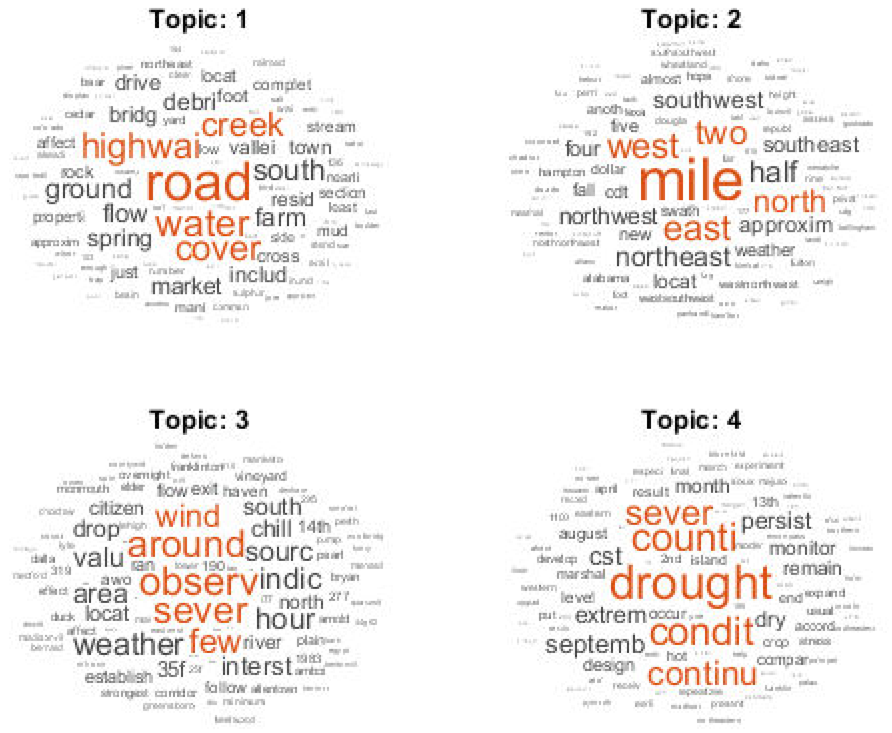
```
numTopics = 60;
mdl = fitlda(bag,numTopics);
```

```
Initial topic assignments sampled in 2.17698 seconds.
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration |  change in | perplexity | concentration | concentration |
|            |  (seconds) |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       2.44 |            |  6.148e+02 |        15.000 |             0 |
|          1 |       9.75 | 1.6206e-01 |  2.511e+02 |        15.000 |             0 |
|          2 |      10.45 | 1.4324e-02 |  2.323e+02 |        15.000 |             0 |
|          3 |      10.54 | 4.5621e-03 |  2.266e+02 |        15.000 |             0 |
|          4 |      10.29 | 3.2075e-03 |  2.227e+02 |        15.000 |             0 |
|          5 |      11.19 | 1.9081e-03 |  2.204e+02 |        15.000 |             0 |
|          6 |      10.15 | 1.1105e-03 |  2.191e+02 |        15.000 |             0 |
|          7 |       9.64 | 1.2089e-03 |  2.177e+02 |       15.000 |            0 |
|          8 |      10.68 | 1.4420e-03 |  2.160e+02 |       15.000 |            0 |
|          9 |      10.48 | 9.7858e-04 |  2.149e+02 |       15.000 |            0 |
|         10 |       9.58 | 1.0064e-03 |  2.137e+02 |       15.000 |            0 |
|         11 |      12.01 | 6.9515e-04 |  2.129e+02 |        7.029 |           18 |
|         12 |      13.82 | 3.7253e-02 |  1.756e+02 |        5.196 |           13 |
|         13 |      10.80 | 1.2478e-02 |  1.648e+02 |        4.611 |           10 |
|         14 |      11.48 | 4.7858e-03 |  1.608e+02 |        4.369 |            8 |
|         15 |      10.89 | 2.6832e-03 |  1.587e+02 |        4.204 |            7 |
|         16 |      11.40 | 2.2533e-03 |  1.569e+02 |        4.052 |            7 |
|         17 |      11.75 | 1.4209e-03 |  1.557e+02 |        3.942 |            7 |
|         18 |      10.65 | 1.4675e-03 |  1.546e+02 |        3.832 |            7 |
|         19 |      12.18 | 1.0584e-03 |  1.538e+02 |        3.750 |            6 |
|         20 |      11.65 | 1.4004e-03 |  1.527e+02 |        3.671 |            6 |
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|         21 |      10.41 | 8.1743e-04 |  1.521e+02 |        3.602 |            6 |
|         22 |      11.94 | 1.0635e-03 |  1.512e+02 |        3.521 |            6 |
|         23 |      12.73 | 1.1552e-03 |  1.504e+02 |        3.446 |            6 |
|         24 |      11.95 | 1.1662e-03 |  1.495e+02 |        3.382 |            6 |
|         25 |      11.28 | 8.5580e-04 |  1.489e+02 |        3.339 |            5 |
|         26 |      11.73 | 7.5594e-04 |  1.483e+02 |        3.294 |            5 |
|         27 |      11.62 | 4.9627e-04 |  1.479e+02 |        3.232 |            6 |
|         28 |      11.62 | 9.2081e-04 |  1.473e+02 |        3.190 |            5 |
|         29 |      10.54 | 9.3369e-04 |  1.466e+02 |        3.144 |            5 |
|         30 |      10.72 | 8.4445e-04 |  1.460e+02 |        3.104 |            5 |
|         31 |      11.05 | 5.4654e-04 |  1.456e+02 |        3.078 |            4 |
|         32 |      12.38 | 4.2853e-04 |  1.452e+02 |        3.030 |            5 |
|         33 |      14.09 | 9.9635e-04 |  1.445e+02 |        2.995 |            5 |
|         34 |      14.46 | 5.2839e-04 |  1.442e+02 |        2.970 |            4 |
|         35 |      14.34 | 7.1177e-04 |  1.436e+02 |        2.924 |            5 |
|         36 |      15.88 | 5.9159e-04 |  1.432e+02 |        2.900 |            4 |
|         37 |      13.42 | 5.4132e-04 |  1.428e+02 |        2.866 |            5 |
|         38 |      11.22 | 7.0649e-04 |  1.423e+02 |        2.833 |            5 |
|         39 |      10.90 | 5.2818e-04 |  1.420e+02 |        2.819 |            3 |
|         40 |      10.29 | 1.1893e-04 |  1.419e+02 |        2.781 |            5 |
=====================================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            |  iteration | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|          41 |       11.52 | 5.5626e-04 |  1.415e+02 |          2.757 |              4 |
|          42 |       11.17 | 5.0465e-04 |  1.411e+02 |          2.728 |              4 |
|          43 |       10.85 | 5.1489e-04 |  1.408e+02 |          2.705 |              4 |
|          44 |       10.67 | 3.0931e-04 |  1.406e+02 |          2.666 |              5 |
|          45 |       11.02 | 3.7067e-04 |  1.403e+02 |          2.635 |              5 |
|          46 |        9.58 | 3.5919e-04 |  1.401e+02 |          2.624 |              3 |
|          47 |        9.56 | 8.5336e-05 |  1.400e+02 |          2.601 |              4 |
=====================================================================================
```

Visualize Topics Using Word Clouds

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

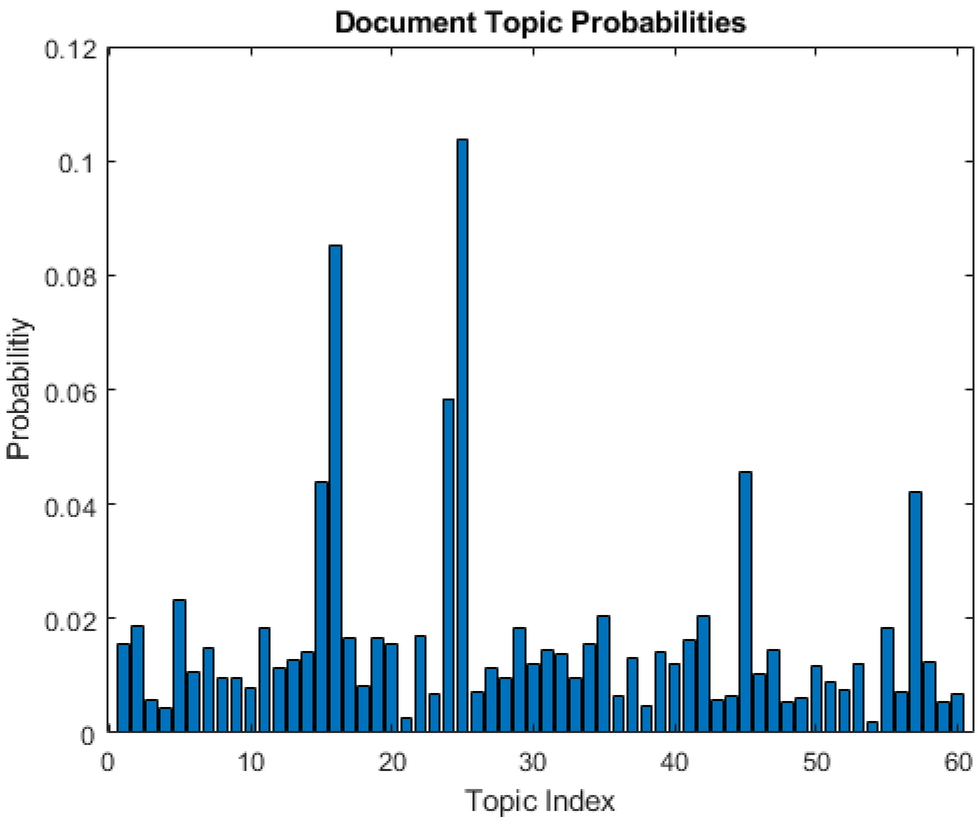
```
figure;
for topicIdx = 1:4
    subplot(2,2,topicIdx)
    wordcloud(mdl,topicIdx);
    title("Topic: " + topicIdx)
end
```



View Mixtures of Topics in Documents

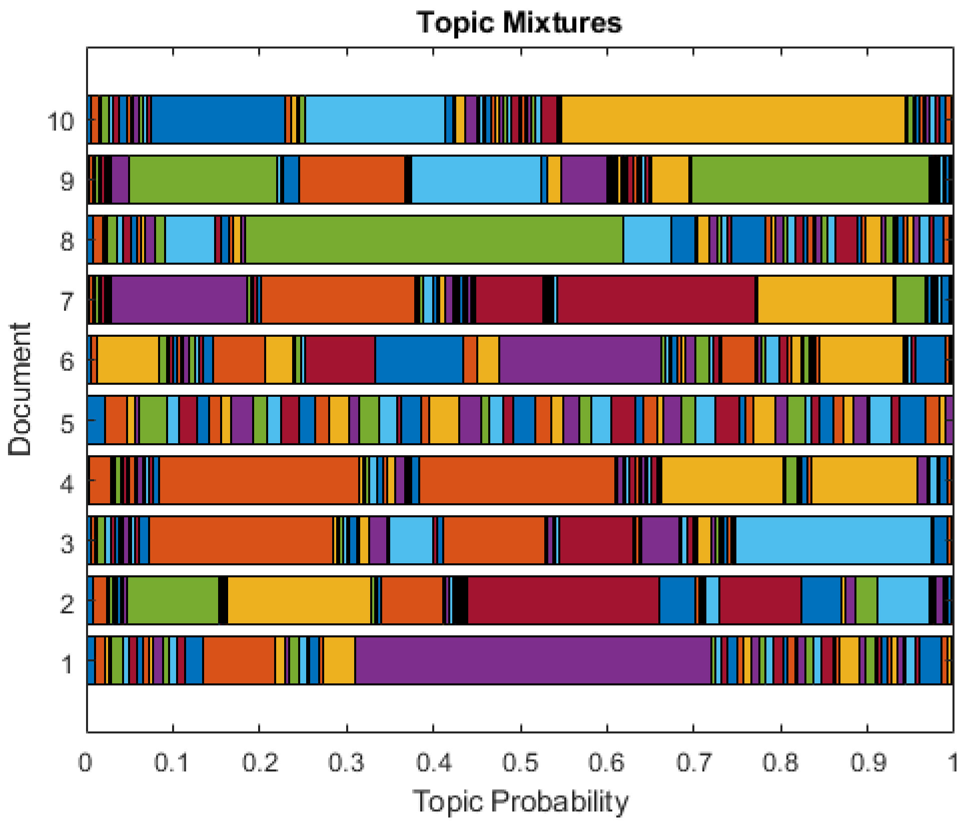
Use `transform` to transform the documents into vectors of topic probabilities.

```
newDocument = tokenizedDocument("A tree is downed outside Apple Hill Drive, Natick");
topicMixture = transform(mdl,newDocument);
figure
bar(topicMixture)
xlabel("Topic Index")
ylabel("Probabilitiy")
title("Document Topic Probabilities")
```



Visualize multiple topic mixtures using stacked bar charts. Visualize the topic mixtures of the first 10 input documents.

```
figure
topicMixtures = transform(mdl,documents(1:10));
barh(topicMixtures(1:10,:),'stacked')
xlim([0 1])
title("Topic Mixtures")
xlabel("Topic Probability")
ylabel("Document")
```



Example Preprocessing Function

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

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

7 Normalize the words using the Porter stemmer using `normalizeWords`.

```
function [documents] = preprocessWeatherNarratives(textData)
% Erase punctuation.
cleanTextData = erasePunctuation(textData);

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

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

% Remove a list of stop words.
documents = removeWords(documents,stopWords);

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

% Normalize the words using the Porter stemmer.
documents = normalizeWords(documents);
end
```

See Also

`bagOfWords` | `fitlda` | `ldaModel` | `tokenizedDocument` | `wordcloud`

Related Examples

- "Choose Number of Topics for LDA Model" on page 1-41
- "Compare LDA Solvers" on page 1-46
- "Analyze Text Data Using Multiword Phrases" on page 1-51
- "Classify Text Data Using Deep Learning" on page 1-68

Choose Number of Topics for LDA Model

This example shows how to decide on a suitable number of topics for a latent Dirichlet allocation (LDA) model.

To decide on a suitable number of topics, you can compare the goodness-of-fit of LDA models fit with varying numbers of topics. You can evaluate the goodness-of-fit of an LDA model by calculating the perplexity of a held-out set of documents. The perplexity indicates how well the model describes a set of documents. A lower perplexity suggests a better fit.

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

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

Extract and Preprocess Text Data

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

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

Tokenize and preprocess the text data using the function `preprocessWeatherNarratives` which is listed at the end of this example.

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

```
ans = 
  5×1 tokenizedDocument:

(1,1)  5 tokens: larg tree down plantersvil nettleton
(2,1) 18 tokens: two feet deep stand water develop street winthrop univers campu inch
(3,1)  9 tokens: nw columbia relai report tree blown down tom hall
(4,1) 10 tokens: media report two tree blown down i40 old fort area
(5,1)  0 tokens:
```

Set aside 10% of the documents at random for validation.

```
numDocuments = numel(documents);
cvp = cvpartition(numDocuments,'HoldOut',0.1);
```

```
documentsTrain = documents(cvp.training);
documentsValidation = documents(cvp.test);
```

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

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

Choose Number of Topics

The goal is to choose a number of topics that mini the perplexity is lowest compared to other numbers of topics. This is not the only consideration: models fit with larger numbers of topics may take longer to converge. To see the effects of the tradeoff, calculate both goodness-of-fit and the fitting time. If the optimal number of topics is high, then you might want to choose a lower value to speed up the fitting process.

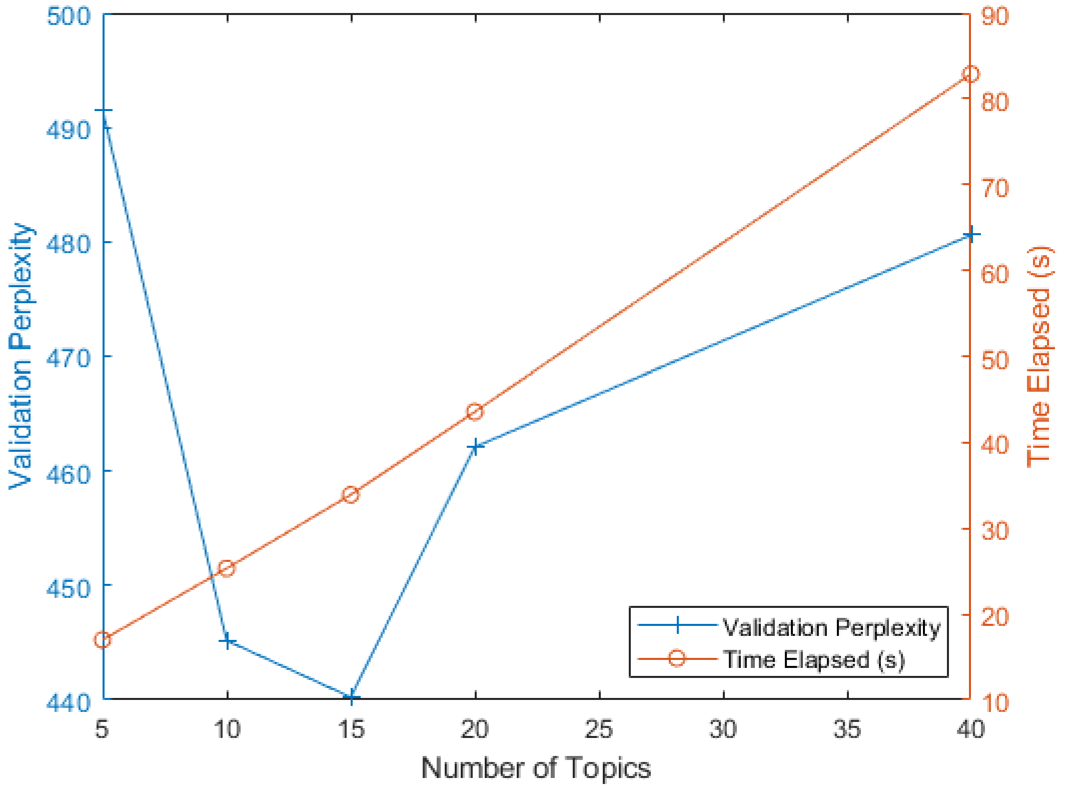
Fit some LDA models for a range of values for the number of topics. Compare the fitting time and the perplexity of each model on the held-out set of test documents. The perplexity is the second output to the `logp` function. To obtain the second ouput without assigning the first output to anything, use the ~ symbol. The fitting time is the `TimeSinceStart` value for the last iteration. This value is in the `History` struct of the `FitInfo` property of the LDA model.

For a quicker fit, specify `'Solver'` to be `'savb'`. To train for more passes of the data, specify `'DataPassLimit'` to be 10. To suppress verbose output, set `'Verbose'` to `0`. This may take a few minutes to run.

```
numTopicsRange = [5 10 15 20 40];
for i = 1:numel(numTopicsRange)
    numTopics = numTopicsRange(i);

    mdl = fitlda(bag,numTopics, ...
        'Solver','savb', ...
        'DataPassLimit',10,...
        'Verbose',0);

    [~,validationPerplexity(i)] = logp(mdl,documentsValidation);
    timeElapsed(i) = mdl.FitInfo.History.TimeSinceStart(end);
end
```

Show the perplexity and elapsed time for each number of topics in a plot. Plot the perplexity on the left axis and the time elapsed on the right axis.

```
figure
yyaxis left
plot(numTopicsRange,validationPerplexity,'+-')
ylabel("Validation Perplexity")

yyaxis right
plot(numTopicsRange,timeElapsed,'o-')
ylabel("Time Elapsed (s)")

legend(["Validation Perplexity" "Time Elapsed (s)"],'Location','southeast')
xlabel("Number of Topics")
```



The plot suggests that fitting a model with 10–20 topics may be a good choice. The perplexity is low compared with the models with different numbers of topics. With this solver, the elapsed time for this many topics is also reasonable. With different solvers, you may find that increasing the number of topics can lead to a better fit, but fitting the model takes longer to converge.

Example Preprocessing Function

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

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

```
function [documents] = preprocessWeatherNarratives(textData)
% Erase punctuation.
cleanTextData = erasePunctuation(textData);

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

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

% Remove a list of stop words.
documents = removeWords(documents,stopWords);

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

% Normalize the words using the Porter stemmer.
```

```
documents = normalizeWords(documents);
end
```

See Also

`bagOfWords` | `fitlda` | `ldaModel` | `logp` | `tokenizedDocument` | `wordcloud`

Related Examples

- “Analyze Text Data Using Topic Models” on page 1-32
- “Compare LDA Solvers” on page 1-46

Compare LDA Solvers

This example shows how to compare latent Dirichlet allocation (LDA) solvers by comparing the goodness of fit and the time taken to fit the model.

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

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

Extract and Preprocess Text Data

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

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

Tokenize and preprocess the text data using the function `preprocessWeatherNarratives` which is listed at the end of this example.

```
documents = preprocessWeatherNarratives(textData);
```

Set aside 10% of the documents at random for validation.

```
numDocuments = numel(documents);
cvp = cvpartition(numDocuments,'HoldOut',0.1);
documentsTrain = documents(cvp.training);
documentsValidation = documents(cvp.test);
```

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

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

Fit and Compare Models

For each of the LDA solvers, fit an LDA model with 60 topics. To distinguish the solvers when plotting the results on the same axes, specify different line properties for each solver.

```
numTopics = 60;
solvers = ["cgs" "avb" "cvb0" "savb"];
lineSpecs = ["+-" "*-" "x-" "o-"];
```

For the validation data, create a bag-of-words model from the validation documents.

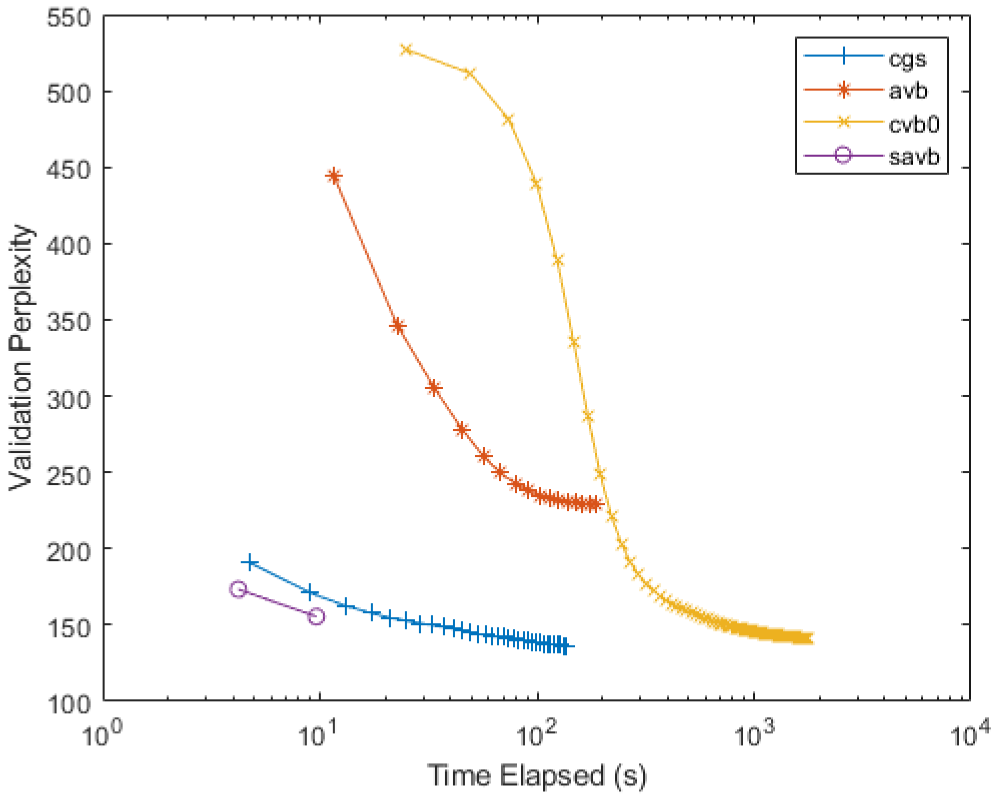
```
validationData = bagOfWords(documentsValidation);
```

For each of the LDA solvers, fit the model, set the initial topic concentration to 1, and specify to not fit the topic concentration parameter. Using the data in the `FitInfo` property of the fitted LDA models, plot the validation perplexity and the time elapsed. Plot the time elapsed in a logarithmic scale. This can take up to an hour to run.

The code for removing NaNs is necessary because of a quirk of the stochastic solver `'savb'`. For this solver, the function evaluates the validation perplexity after each pass of the data. The function does not evaluate the validation perplexity for each iteration (mini-batch) and reports NaNs in the `FitInfo` property. To plot the validation perplexity, remove the NaNs from the reported values.

```
figure
for i = 1:numel(solvers)
    solver = solvers(i);
    lineSpec = lineSpecs(i);

    mdl = fitlda(bag,numTopics, ...
        'Solver',solver, ...
        'InitialTopicConcentration',1, ...
        'FitTopicConcentration',false, ...
        'ValidationData',validationData, ...
        'Verbose',0);

    history = mdl.FitInfo.History;

    timeElapsed = history.TimeSinceStart;
    validationPerplexity = history.ValidationPerplexity;

    % Remove NaNs.
    idx = isnan(validationPerplexity);
    timeElapsed(idx) = [];
    validationPerplexity(idx) = [];

    semilogx(timeElapsed,validationPerplexity,lineSpec)
    hold on
end
```

```
hold off
xlabel("Time Elapsed (s)")
ylabel("Validation Perplexity")
legend(solvers)
```



For the stochastic solver `"savb"`, the function, by default, passes through the training data once. To process more passes of the data, set `'DataPassLimit'` to a larger value (the default value is 1). For the batch solvers (`"cgs"`, `"avb"`, and `"cvb0"`), to reduce the number of iterations used to fit the models, set the `'IterationLimit'` option to a lower value (the default value is 100).

A lower validation perplexity suggests a better fit. Usually, the solvers `"savb"` and `"cgs"` converge quickly to a good fit. The solver `"cvb0"` might converge to a better fit, but it can take much longer to converge.

For the `FitInfo` property, the `fitlda` function estimates the validation perplexity from the document probabilities at the maximum likelihood estimates of the per-document topic probabilities. This is usually quicker to compute, but can be less accurate than other methods. To calculate more accurate values, calculate the validation perplexity using the `logp` function. This will take longer to run. For an example showing how to compute the perplexity using `logp`, see .

Example Preprocessing Function

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

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

```
function [documents] = preprocessWeatherNarratives(textData)
% Erase punctuation.
cleanTextData = erasePunctuation(textData);

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

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

% Remove a list of stop words.
documents = removeWords(documents,stopWords);

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

```
% Normalize the words using the Porter stemmer.
documents = normalizeWords(documents);
end
```

See Also

`bagOfWords` | `fitlda` | `ldaModel` | `logp` | `tokenizedDocument` | `wordcloud`

Related Examples

- “Analyze Text Data Using Topic Models” on page 1-32
- “Choose Number of Topics for LDA Model” on page 1-41

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 = 5x1 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. Erase punctuation using `erasePunctuation`.
2. Convert the text data to lowercase using `lower`.
3. Tokenize the text using `tokenizedDocument`.
4. Remove a list of stop words (such as "and", "of", and "the") using `removeWords` and `stopWords`.
5. Remove words with 2 or fewer characters using `removeShortWords`.
6. Remove words with 15 or more characters using `removeLongWords`.
7. Normalize the words using the Porter stemmer using `normalizeWords`.

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

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

```
ans = 
  5x1 tokenizedDocument:

(1,1)  5 tokens: larg tree down plantersvil nettleton
(2,1) 18 tokens: two feet deep stand water develop street winthrop univers campu inch
(3,1)  9 tokens: nw columbia relai report tree blown down tom hall
(4,1) 10 tokens: media report two tree blown down i40 old fort area
(5,1)  8 tokens: few tree limb greater inch 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`.

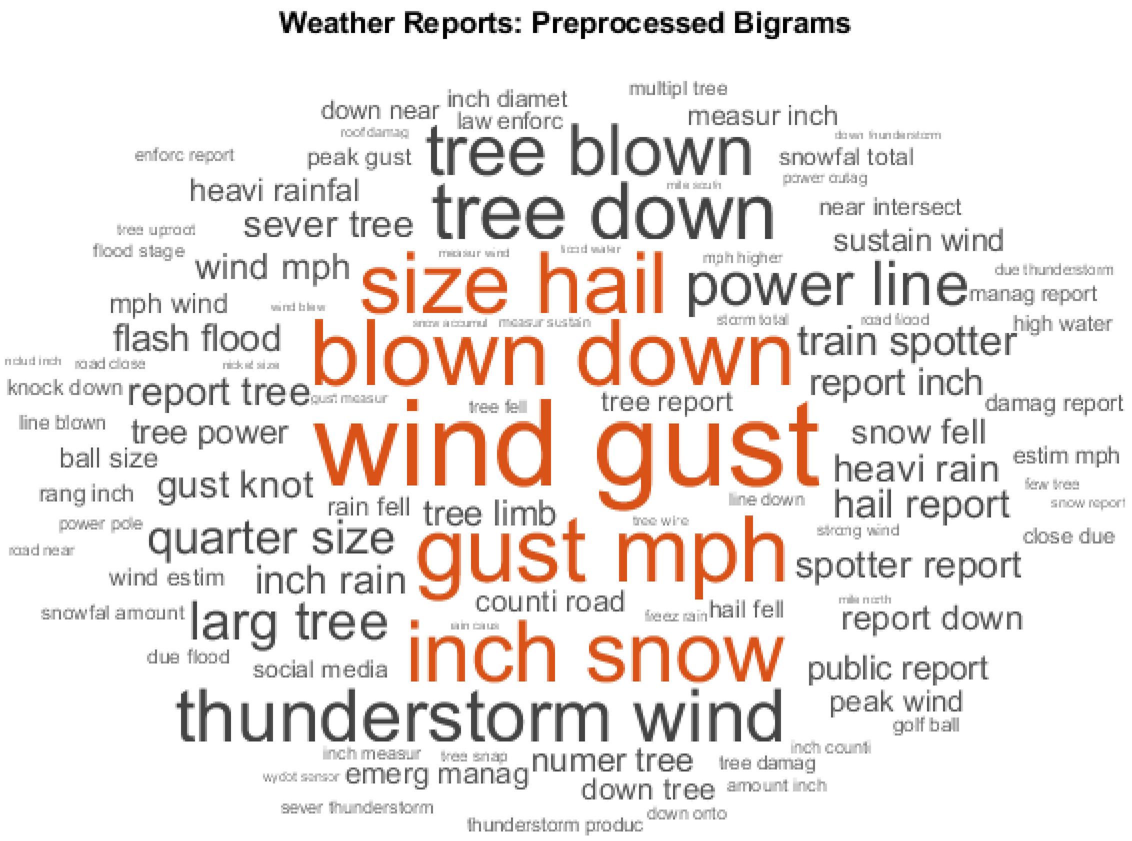
To count the n-grams of length 2 (bigrams), use `bagOfNgrams` with the default options.

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

```
bag = 
  bagOfNgrams with properties:

          Counts: [28138x114788 double]
      Vocabulary: [1x17815 string]
          Ngrams: [114788x2 string]
    NgramLengths: 2
       NumNgrams: 114788
    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.

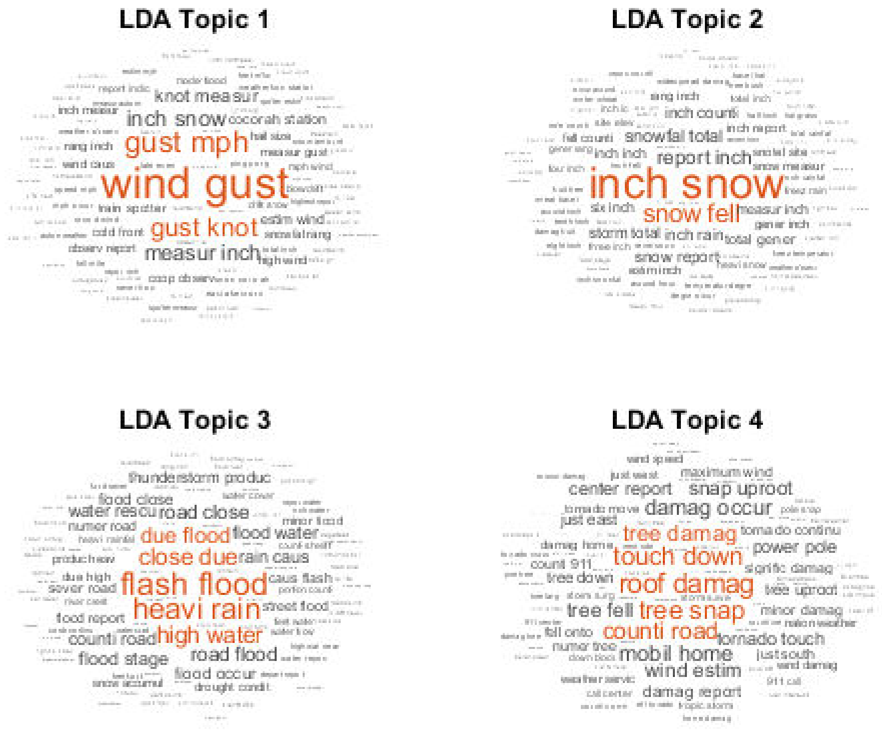
Create an LDA topic model with 10 topics using `fitlda`. The function fits an LDA model by treating the n-grams as single words.

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

```
Initial topic assignments sampled in 1.68128 seconds.
==============================================================================
| Iteration  |  Time per  |  Relative  |  Training  |     Topic     |     Topic     |
|            | iteration  | change in  | perplexity | concentration | concentration |
|            | (seconds)  |   log(L)   |            |               |   iterations  |
=====================================================================================
|          0 |       6.49 |            |  1.919e+04 |         2.500 |             0 |
|          1 |       8.56 | 7.0045e-02 |  1.006e+04 |         2.500 |             0 |
|          2 |       8.63 | 2.2377e-03 |  9.860e+03 |         2.500 |             0 |
|          3 |       9.68 | 3.9570e-04 |  9.824e+03 |         2.500 |             0 |
|          4 |       9.71 | 4.1796e-04 |  9.786e+03 |         2.500 |             0 |
|          5 |       9.64 | 1.4536e-05 |  9.785e+03 |         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 cooccuring 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`.

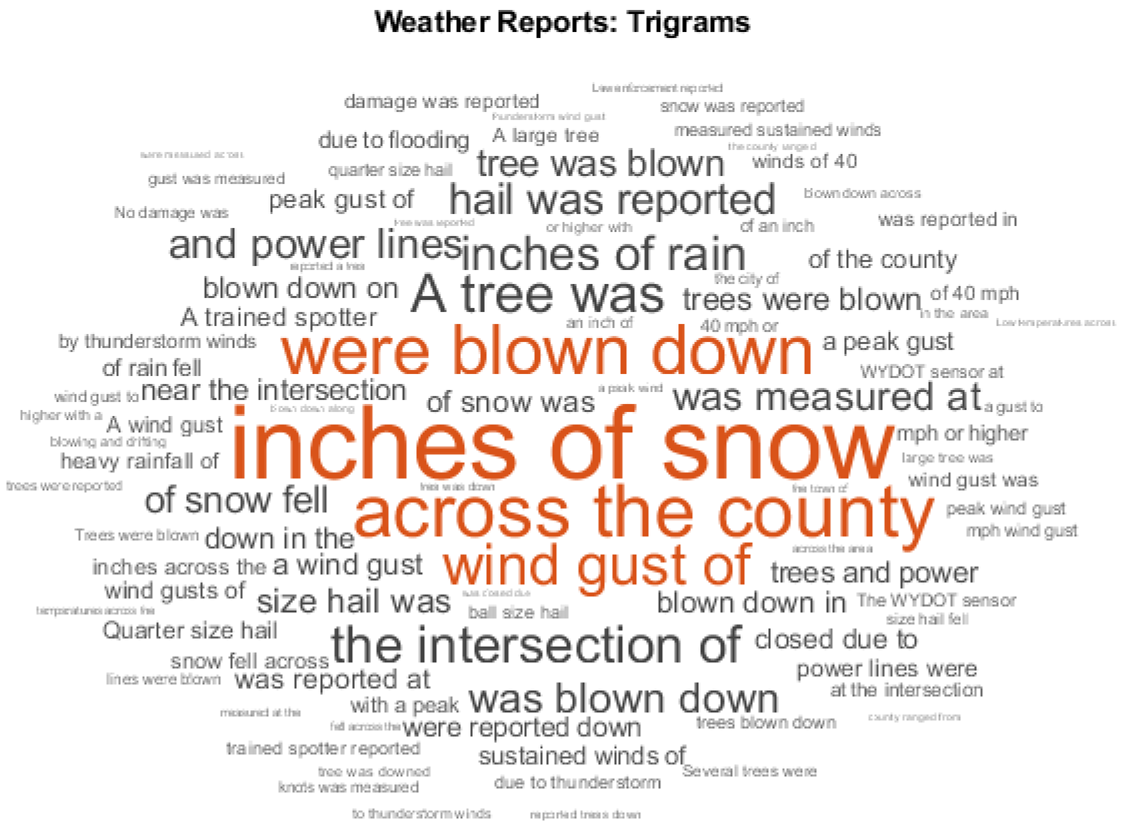
```
cleanTextData = erasePunctuation(textData);
documents = tokenizedDocument(cleanTextData);
```

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:

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

```
function documents = preprocessWeatherNarratives(textData)
% Erase punctuation.
cleanTextData = erasePunctuation(textData);

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

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

```
% Remove a list of stop words.
documents = removeWords(documents,stopWords);

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

% Normalize the words using the Porter stemmer.
documents = normalizeWords(documents);

end
```

See Also

`bagOfNgrams` | `fitlda` | `ldaModel` | `tokenizedDocument` | `topkngrams` | `wordcloud`

Related Examples

- "Analyze Text Data Using Topic Models" on page 1-32
- "Visualize Text Data Using Word Clouds" on page 1-26
- "Classify Text Data Using Deep Learning" on page 1-68

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 "king is to queen as man is to woman". In vector form, this relationship is $king - man + woman = queen$.

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

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

Read the example word embedding. This model was derived by analyzing text from Wikipedia.

```
filename = "exampleWordEmbedding.vec";
emb = readWordEmbedding(filename)
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 50
    Vocabulary: [1x9999 string]
```

Explore the word embedding using `word2vec` and `vec2word`. Convert the words *king*, *man*, and *woman* to vectors using `word2vec`.

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

Compute the vector given by `king - man + woman`. This vector encapsulates the semantic meaning of the word *king*, without the semantics of the word *man*, and also includes the semantics of the word *woman*.

```
vec = king - man + woman
```

```
vec = 1x50 single row vector

   -0.9633    0.2275    0.9614    2.1593   -1.0541   -4.7783   -2.5908   -1.0410   -0.2
```

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

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

```
word = 
"queen"
```

Create 2-D Text Scatter Plot

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

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

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

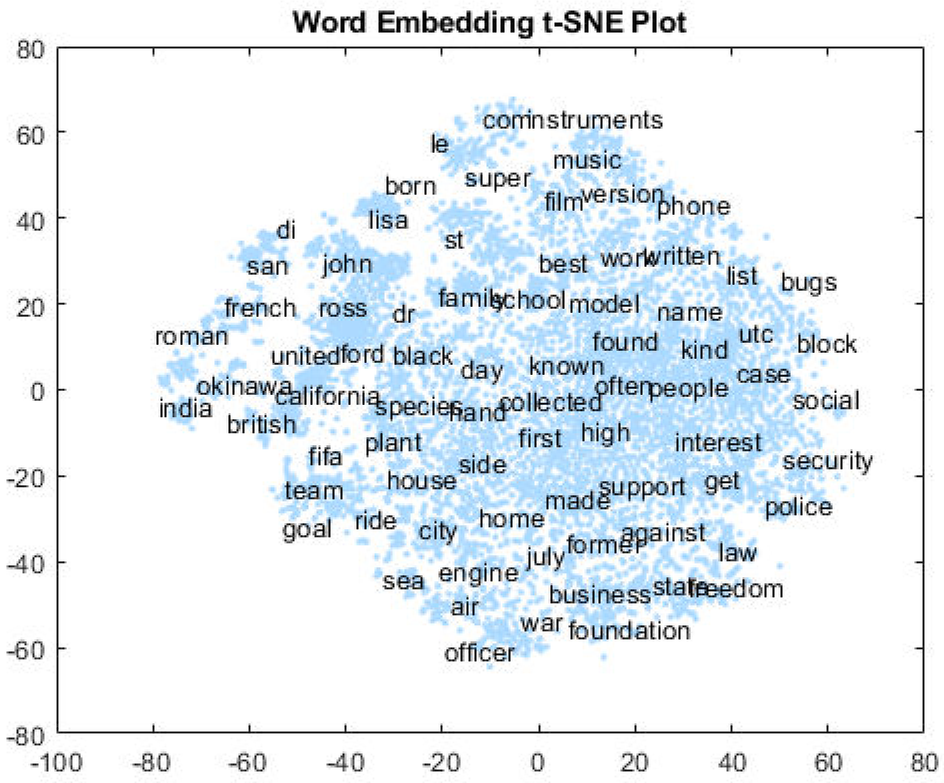
```
ans = 1×2

        9999          50

```

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 you can set the `'Verbose'` name-value pair to 1.

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

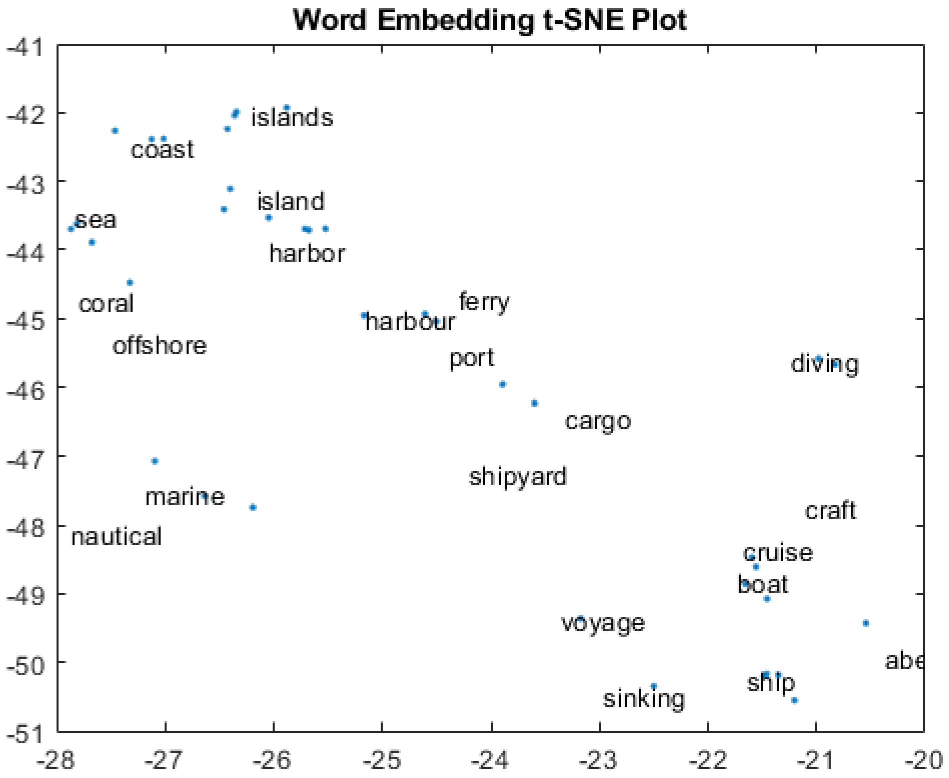
Plot the words at the coordinates specified by `XY` in a 2-D text scatter plot. For readability, `textscatter`, by default, does not display all of the input words and displays markers instead.

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



Zoom in on a section of the plot.

```
xlim([-28 -20])
ylim([-51 -41])
```



Create 3-D Text Scatter Plot

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

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

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

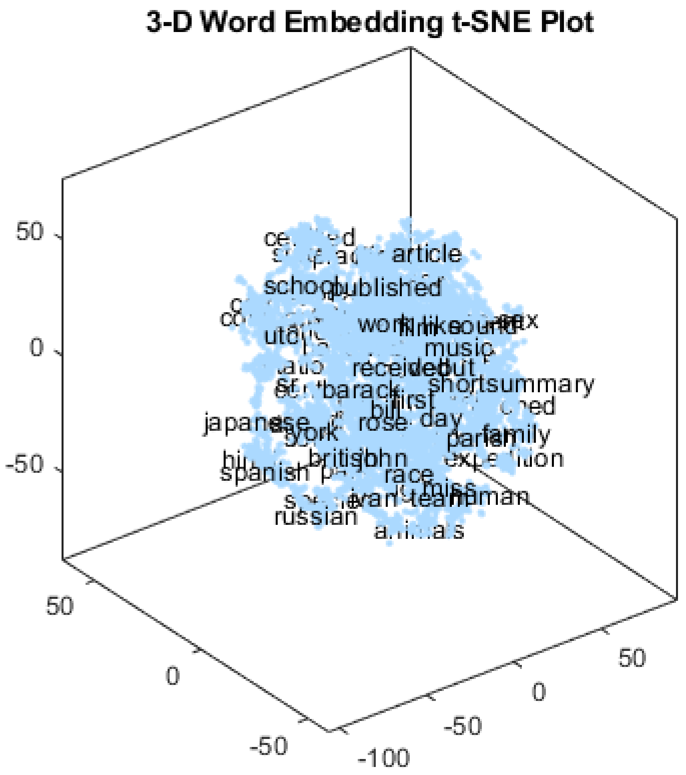
```
ans = 1×2

        9999           50
```

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);
```

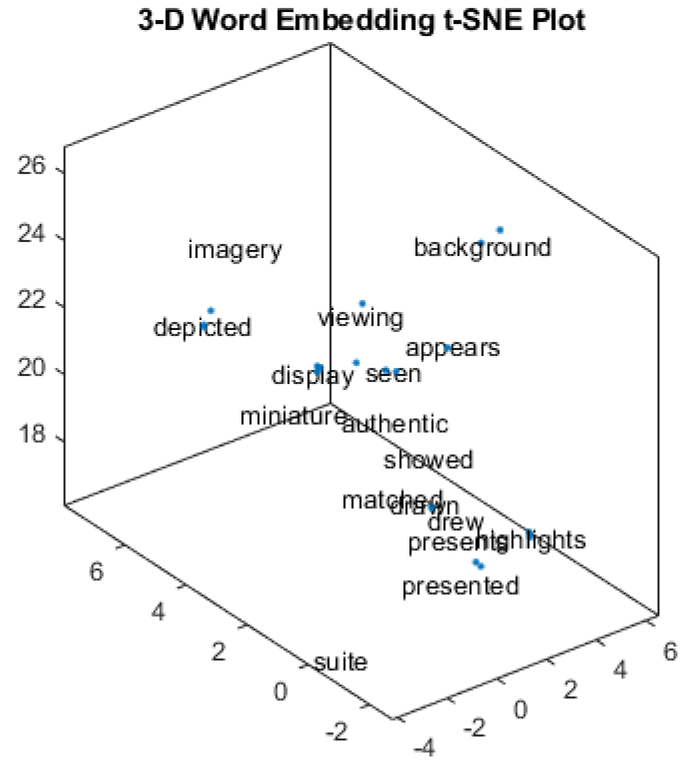
Plot the words at the coordinates specified by XYZ in a 3-D text scatter plot.

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



Zoom in on a section of the plot.

```
xlim([-4.2 6.5])
ylim([-2.72 7.99])
zlim([16.10 26.81])
```



Perform Cluster Analysis

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

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

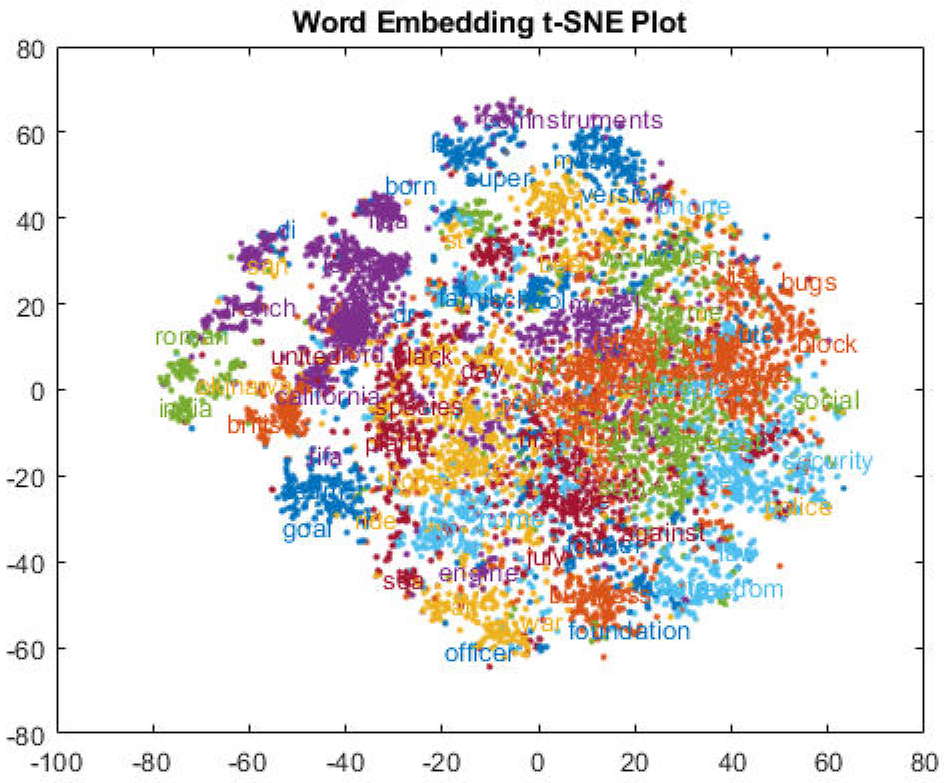
```
ans = 1×2

        9999          50
```

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")
```



See Also

`readWordEmbedding` | `textscatter` | `textscatter3` | `word2vec` | `wordEmbedding`

Related Examples

- "Extract Text Data from Files" on page 1-2
- "Prepare Text Data for Analysis" on page 1-16
- "Visualize Text Data Using Word Clouds" on page 1-26
- "Classify Text Data Using Deep Learning" on page 1-68

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, you can 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 use an LSTM network for text data, you must first convert the text data into numeric sequences. You can achieve this using word embeddings. 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*.

This example shows how to train a word-embedding to convert text data to sequences of numeric vectors and train an LSTM network for text classification. There are four steps in training and using the LSTM network:

- Import and preprocess the data.
- Convert the words to numeric vectors by training a word embedding.
- Create and train an LSTM network using the sequences of word vectors.
- 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 string arrays, 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.

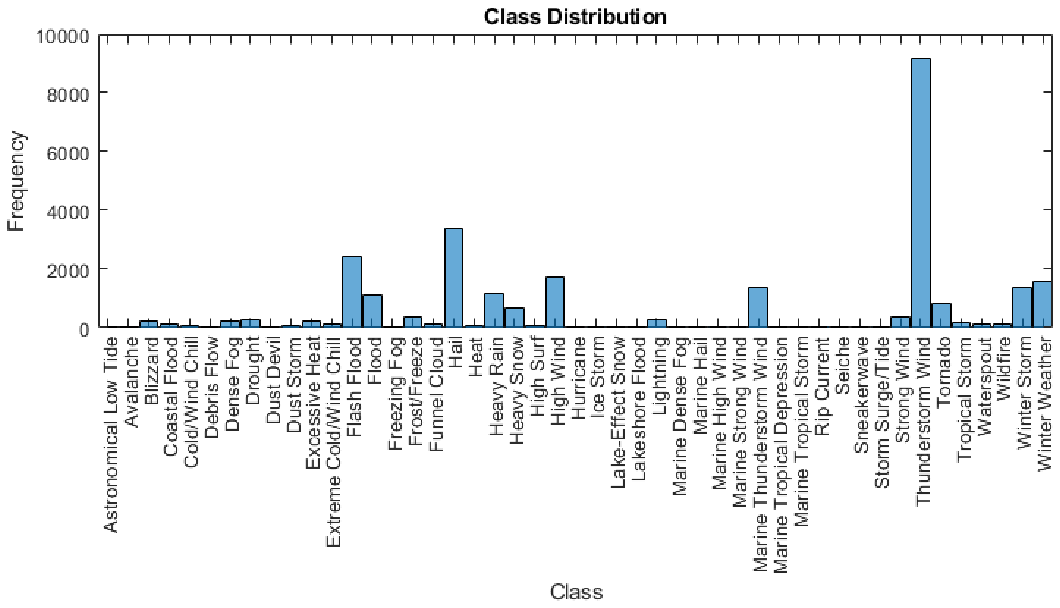
```
idxEmpty = strlength(data.event_narrative) == 0;
data(idxEmpty,:) = [];
```

Convert the labels in the `event_type` column of the table 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 suboptimal result. To prevent this problem, 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.

```
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 classes from the categorical data.

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

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

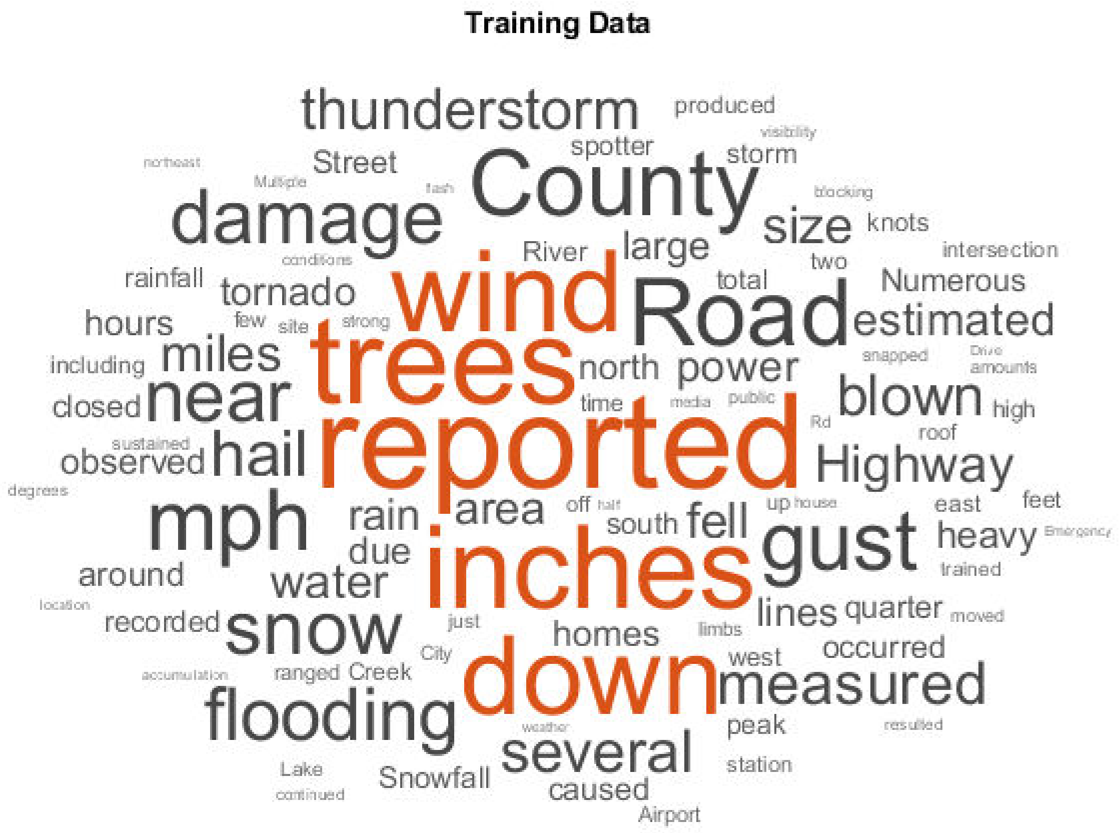
```
cvp = cvpartition(data.event_type,'Holdout',0.1);
dataTrain = data(training(cvp),:);
dataTest = data(test(cvp),:);
```

Extract the text data and labels from the partitioned tables.

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

Visualize the training text data using a word cloud.

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



Preprocess Text Data

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

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

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 street on the wi
(3,1) 13 tokens: nws columbia relayed a report of trees blown down along tom hall st
(4,1) 13 tokens: media reported two trees blown down along i40 in the old fort area
(5,1) 14 tokens: a few tree limbs greater than 6 inches down on hwy 18 in roseland
```

Train 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.*

Train a word embedding with dimension 100. To train for longer, specify the number of training epochs to be 50. An epoch corresponds to one pass through the training data. To suppress the verbose output, set `'Verbose'` to 0. This can take several minutes to run.

```
embeddingDimension = 100;
embeddingEpochs = 50;

emb = trainWordEmbedding(documentsTrain, ...
    'Dimension',embeddingDimension, ...
    'NumEpochs',embeddingEpochs, ...
    'Verbose',0)
```

```
emb = 
  wordEmbedding with properties:

     Dimension: 100
    Vocabulary: [1×4996 string]
```

Convert Document to Sequences

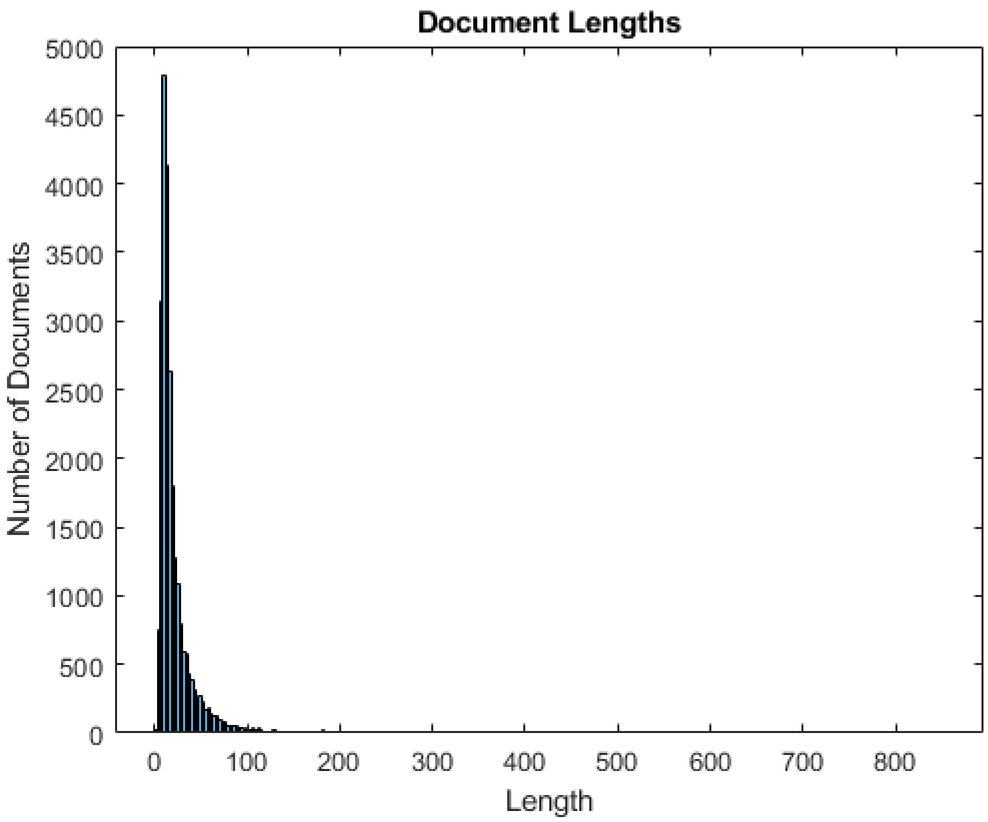
To input the documents into an LSTM network, convert the documents into sequences of word vectors.

When training the network, the software creates mini-batches of sequences of the same length by padding, truncating, or splitting the input data. The `trainingOptions` function provides options to pad and truncate input sequences, however, these options are not well suited for sequences of word vectors. Instead, you must pad and truncate the sequences manually. If you *left-pad* and truncate the sequences of word vectors, then the training might improve.

The first conversion step is to 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. Truncate the training documents to have length 75 using `docfun`. The anonymous function inputted to `docfun` takes string array input and outputs the first 75 elements.

```
sequenceLength = 75;
documentsTruncatedTrain = docfun(@(words) words(1:min(sequenceLength,end)),documentsTra
```

Convert the documents to sequences of word vectors. To convert the training documents into a cell array of sequences, use the example function `doc2sequence`, shown at the end of this example. The columns of each sequence are the word vectors. If you have Parallel Computing Toolbox™ installed, then the function loops through the documents in parallel. Otherwise, the function loops through the documents in series and can take a few minutes to run.

```
XTrain = doc2sequence(emb,documentsTruncatedTrain);
```

```
Starting parallel pool (parpool) using the 'local' profile ...
connected to 6 workers.
```

```
XTrain(1:5)
```

```
ans = 1×5 cell array
    {100×5 single}    {100×37 single}    {100×13 single}    {100×13 single}    {100×14
```

Pad with zeros the documents with fewer tokens than the fixed length. To pad sequences of word vectors for LSTM networks, you must *left-pad* the sequences. The sequence padding option for LSTM networks, by default, *right-pads* the sequences, so you must do this manually.

Apply the example function `leftPad`, shown at the end of this example, to each of the sequences in `XTrain`. This function left-pads the sequences with zeros so that they have the same length.

```
for i = 1:numel(XTrain)
    XTrain{i} = leftPad(XTrain{i},sequenceLength);
end
XTrain(1:5)
```

```
ans = 1×5 cell array
    {100×75 single}    {100×75 single}    {100×75 single}    {100×75 single}    {100×75
```

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 be the dimension of the word embedding. Next, include an LSTM layer and specify the output size to be 180. To use the LSTM layer for a sequence-to-label classification problem, set the output mode to be `'last'`. Finally, add a fully connected layer with the same size as the number of classes, a softmax layer, and a classification layer.

```
inputSize = embeddingDimension;
outputSize = 180;
numClasses = numel(categories(YTrain));

layers = [ ...
    sequenceInputLayer(inputSize)
    lstmLayer(outputSize,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer]
```

```
layers = 
  5x1 Layer array with layers:

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

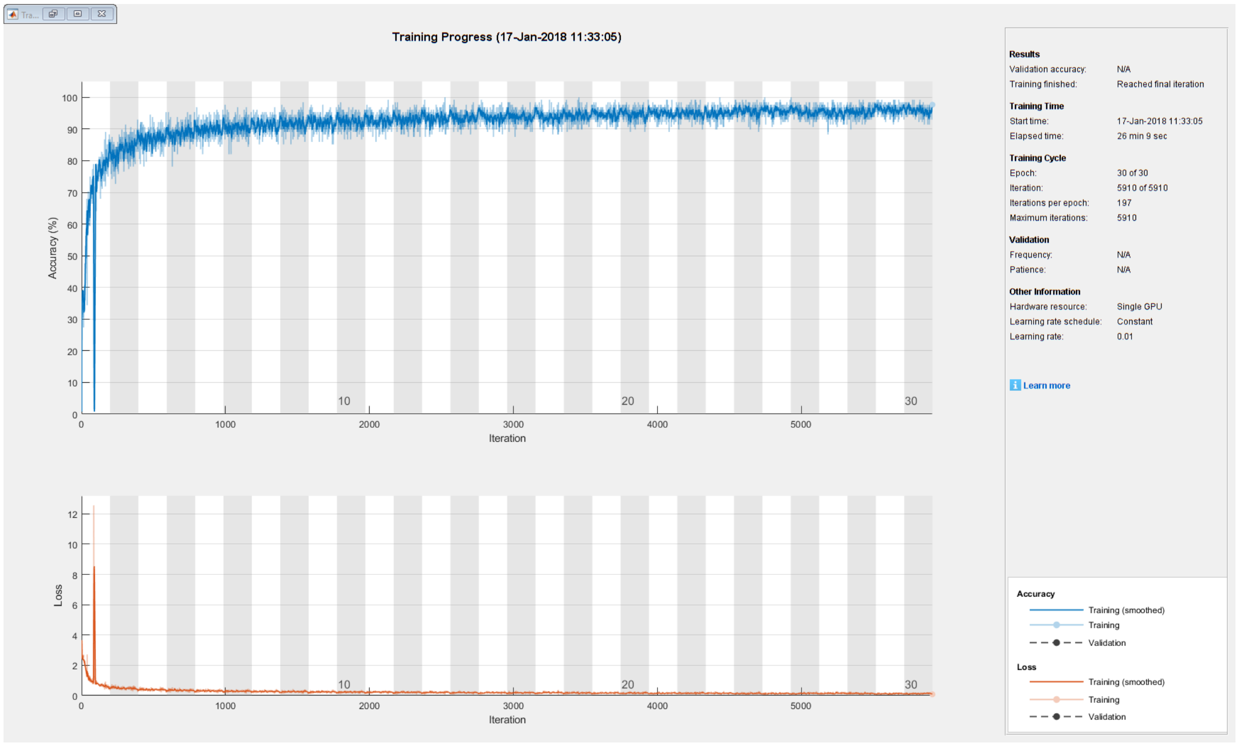
Specify the training options. Specify the solver to be `'adam'`, and the gradient threshold to be 1. Set the initial learn rate to be 0.01. To monitor the training progress, set the `'Plots'` option to `'training-progress'`. To suppress verbose output, set `'Verbose'` to 0.

By default, `trainNetwork` uses a GPU if one is available (requires Parallel Computing Toolbox™ and a CUDA® enabled GPU with compute capability 3.0 or higher). Otherwise, it uses the CPU. To specify the execution environment manually, use the `'ExecutionEnvironment'` name-value pair argument of `trainingOptions`. Training on a CPU can take significantly longer than training on a GPU.

```
options = trainingOptions('adam', ...
    'GradientThreshold',1, ...
    'InitialLearnRate',0.01, ...
    'Plots','training-progress', ...
    'Verbose',0);
```

Train the LSTM network using the `trainNetwork` function.

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



Results
Validation accuracy: N/A
Training finished: Reached final iteration

Training Time
Start time: 17-Jan-2018 11:33:05
Elapsed time: 26 min 9 sec

Training Cycle
Epoch: 30 of 30
Iteration: 5910 of 5910
Iterations per epoch: 197
Maximum iterations: 5910

Validation
Frequency: N/A
Patience: N/A

Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.01

Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

Test LSTM Network

To test the LSTM network, first prepare the test data in the same way as the training data. Then make predictions on the preprocessed test data using the trained LSTM network `net`.

Preprocess the test data using the same steps as the training documents.

```
textDataTest = erasePunctuation(textDataTest);
textDataTest = lower(textDataTest);
documentsTest = tokenizedDocument(textDataTest);
```

Convert the test documents to sequences using the same steps as the training documents.

```
documentsTruncatedTest = docfun(@(words) words(1:min(sequenceLength,end)),documentsTest
XTest = doc2sequence(emb,documentsTruncatedTest);
for i=1:numel(XTest)
    XTest{i} = leftPad(XTest{i},sequenceLength);
```

```
end
XTest(1:5)
```

```
ans = 1×5 cell array
    {100×75 single}    {100×75 single}    {100×75 single}    {100×75 single}    {100×75
```

Classify the test documents using the trained LSTM network.

```
YPred = classify(net,XTest);
```

Calculate the classification accuracy. The accuracy is the proportion of labels that the network predicts correctly.

```
accuracy = sum(YPred == YTest)/numel(YPred)
```

```
accuracy = 0.8837
```

Predict Using New Data

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

```
reportsNew = [ ...
    "Lots of water damage to computer equipment inside the office."
    "A large tree is downed and blocking traffic outside Apple Hill."
    "Damage to many car windshields in parking lot."
    ];
```

Preprocess the text data using the same steps as the training documents.

```
reportsNew = lower(reportsNew);
reportsNew = erasePunctuation(reportsNew);
documentsNew = tokenizedDocument(reportsNew);
```

Convert the text data to sequences using the `doc2sequence` and `leftPad` example functions. Specify the sequence length to be the same as the training data.

```
documentsTruncatedNew = docfun(@(words) words(1:min(sequenceLength,end)),documentsNew)
XNew = doc2sequence(emb,documentsTruncatedNew);
for i=1:numel(XNew)
    XNew{i} = leftPad(XNew{i},sequenceLength);
end
```

Classify the new sequences using the trained LSTM network.

```
[labelsNew,score] = classify(net,XNew);
```

Show the weather reports with their predicted labels.

```
[reportsNew string(labelsNew)]
```

```
ans = 3×2 string array
    "lots of water damage to computer equipment inside the office"      "Flash Flood"
    "a large tree is downed and blocking traffic outside apple hill"    "Thunderstorm W
    "damage to many car windshields in parking lot"                     "Hail"
```

View the top three predictions and their scores for the first report. Sort the prediction scores and select the top three values.

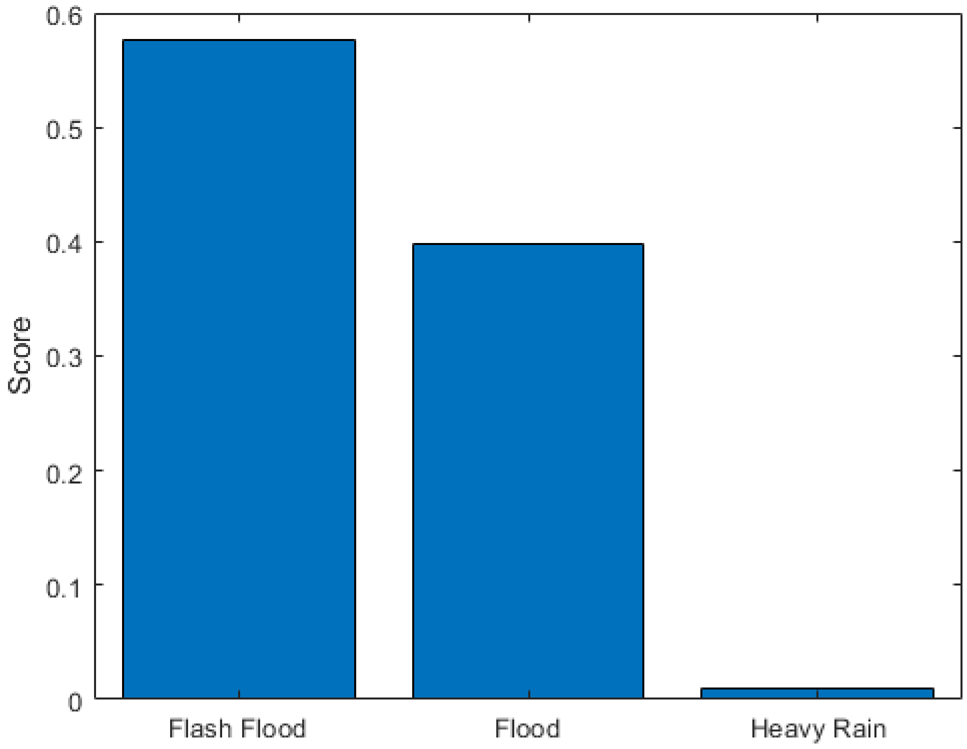
```
[scoreTop,idxTop] = maxk(score(1,:),3);
```

Get the class names from the classification output layer (the last layer) of the LSTM network.

```
classNames = net.Layers(end).ClassNames;
```

Plot the top three classes with their scores in a bar chart.

```
classNamesTop = categorical(classNames(idxTop));
figure
bar(classNamesTop,scoreTop)
ylabel("Score")
```



This bar chart compares the top prediction made by the network, with the two next highest predictions.

Example Functions

The function `doc2sequence` converts documents into a cell array of sequences, where the columns of each sequence are the word vectors. If you have Parallel Computing Toolbox installed, then the `parfor` loop runs in parallel. Otherwise, the loop runs in serial.

```
function C = doc2sequence(emb,documents)

parfor i = 1:numel(documents)
    words = string(documents(i));
    idx = ~ismember(emb,words);
    words(idx) = [];
    C{i} = word2vec(emb,words)';
end

end
```

The function `leftPad` pads matrix `M` with zeros on the left so that it has `N` columns.

```
function MPadded = leftPad(M,N)

[dimension,sequenceLength] = size(M);
paddingLength = N-sequenceLength;
MPadded = [zeros(dimension,paddingLength) M];

end
```

See Also

`docfun` | `lstmLayer` | `sequenceInputLayer` | `textscatter` | `tokenizedDocument` | `trainNetwork` | `trainWordEmbedding` | `trainingOptions`

Related Examples

- "Extract Text Data from Files" on page 1-2
- "Prepare Text Data for Analysis" on page 1-16
- "Analyze Text Data Using Multiword Phrases" on page 1-51
- "Analyze Text Data Using Topic Models" on page 1-32
- "Sequence Classification Using Deep Learning" (Neural Network Toolbox)
- "Deep Learning in MATLAB" (Neural Network Toolbox)