Sentiment Processing

- get sentiment info from textual data

Outline

- Analyze Sentiment in Text
  - vaderSentimentScores
- Generate Domain Specific Sentiment Lexicon
  - trainWordEmbedding
- Train a Sentiment Classifier
  - fitcsvm

Analyze Sentiment in Text

- This example shows how to use the Valence Aware Dictionary and sEntiment Reasoner (VADER) algorithm for sentiment analysis.
- The VADER algorithm uses a list of annotated words (the sentiment lexicon), where each word has a corresponding sentiment score. The VADER algorithm also utilizes word lists that modify the scores of proceeding words in the text:

Analyze Sentiment in Text

- Boosters – words or n-grams that boost the sentiment of proceeding tokens. For example, words like "absolutely" and "amazingly".
- Dampeners – words or n-grams that dampen the sentiment of proceeding tokens. For example, words like "hardly" and "somewhat".
- Negations – words that negate the sentiment of proceeding tokens. For example, words like "not" and "isn't".

Load Data

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

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

Load Data

head(tbl)

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

Load Data

- Create an array of tokenized documents from the text data and view the first few documents.

```
str = tbl.TextData;
documents = tokenizedDocument(str);
documents(1:4)
ans =
 4x1 tokenizedDocument:
 11 tokens: Happy anniversary ! ❤ Next stop : Paris ! ✈ #vacation
 16 tokens: Haha , BBQ on the beach , engage smug mode !   ❤  #vacation
 9 tokens: getting ready for Saturday night  #yum #weekend
 13 tokens: Say it with me - I NEED A #VACATION ! ! ! ☹
```

Evaluate Sentiment

- Evaluate the sentiment of the tokenized documents using the vaderSentimentLexicon function. Scores close to 1 indicate positive sentiment, scores close to -1 indicate negative sentiment, and scores close to 0 indicate neutral sentiment.

```
compoundScores = vaderSentimentScores(documents);
```

- View the scores of the first few documents.

```
compoundScores(1:4)
```

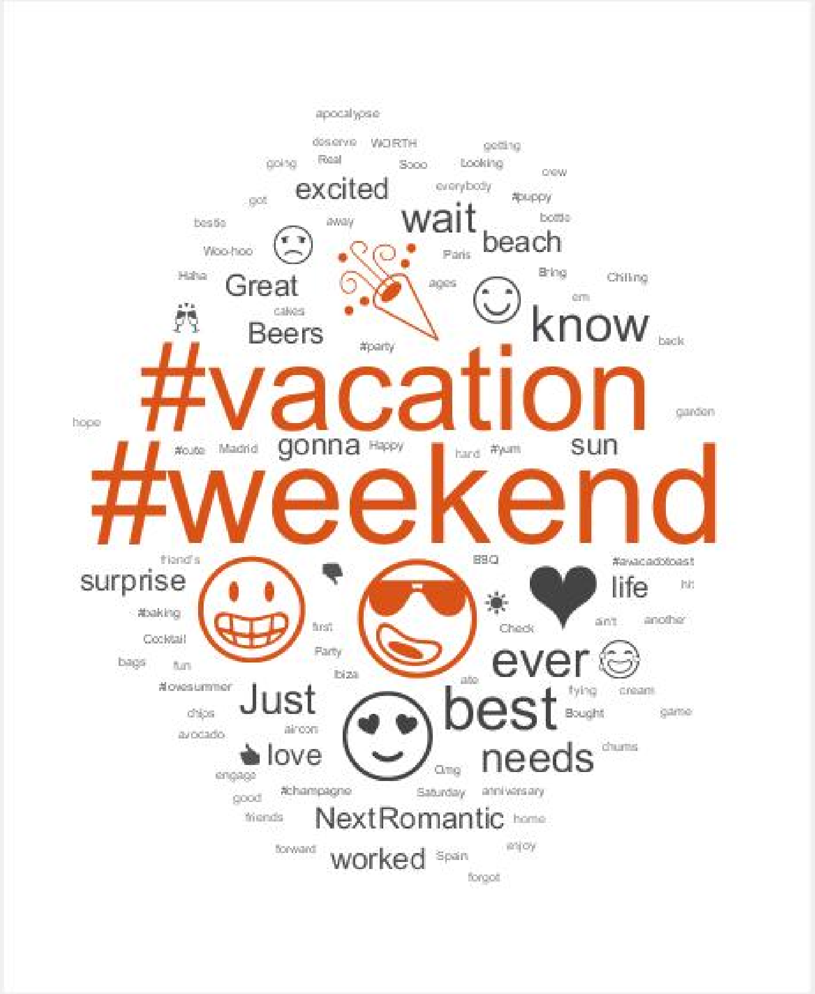
```
ans = 4×1
 0.4738
 0.9348
 0.6705
 -0.5067
```

Evaluate Sentiment

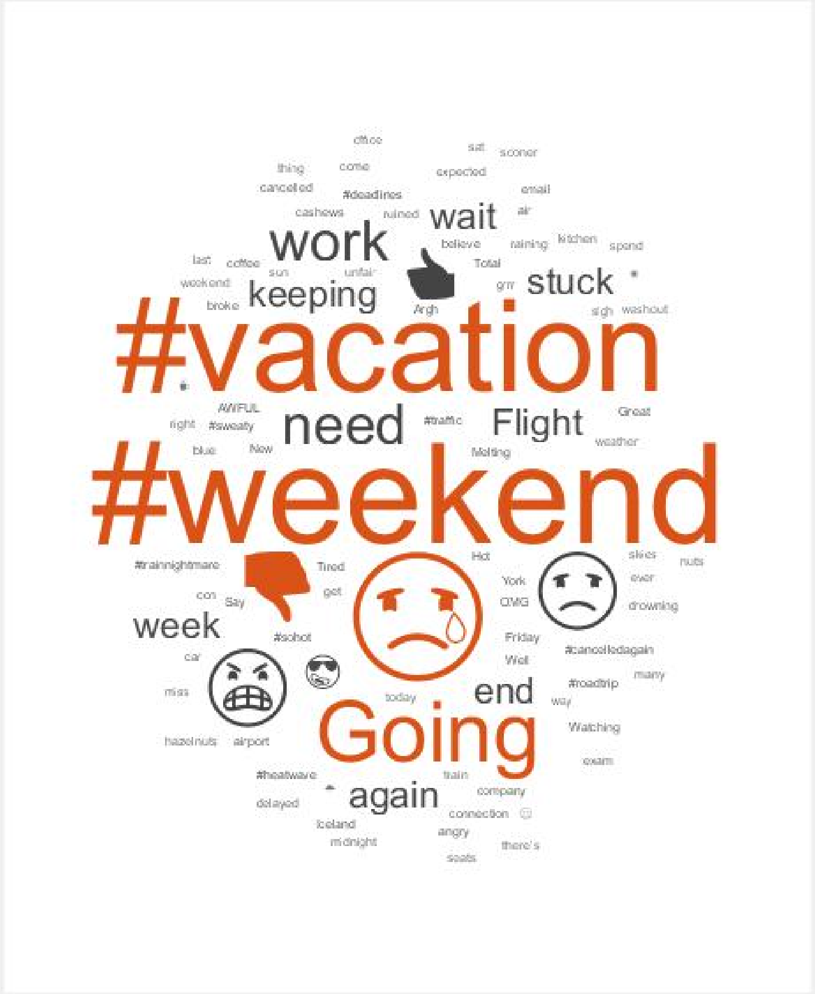
- Visualize the text with sentiment in word clouds.

```
idx = compoundScores > 0;
strPositive = str(idx);
strNegative = str(~idx);
figure
subplot(1,2,1)
wordcloud(strPositive);
title("Positive Sentiment")
subplot(1,2,2)
wordcloud(strNegative);
title("Negative Sentiment")
```

Positive Sentiment



Negative Sentiment



Generate Domain Specific Sentiment Lexicon

- This example shows how to generate a lexicon for sentiment analysis using 10-K and 10-Q financial reports.
- Sentiment analysis allows you to automatically summarize the sentiment in a given piece of text. For example, assign the pieces of text "This company is showing strong growth." and "This other company is accused of misleading consumers." with positive and negative sentiment, respectively. Also, for example, to assign the text "This company is showing extremely strong growth." a stronger sentiment score than the text "This company is showing strong growth."

Generate Domain Specific Sentiment Lexicon

- Sentiment analysis algorithms such as VADER rely on annotated lists of words called sentiment lexicons. For example, VADER uses a sentiment lexicon with words annotated with a sentiment score ranging from -1 to 1, where scores close to 1 indicate strong positive sentiment, scores close to -1 indicate strong negative sentiment, and scores close to zero indicate neutral sentiment.

Generate Domain Specific Sentiment Lexicon

- If the sentiment lexicon used by the vaderSentimentScores function does not suit the data you are analyzing, for example, if you have a domain-specific data set like medical or engineering data, then you can generate your own custom sentiment lexicon using a small set of seed words.

Generate Domain Specific Sentiment Lexicon

- This example shows how to generate a sentiment lexicon given a collection of seed words:
  - Train a word embedding that models the similarity between words using the training data.
  - Create a simplified graph representing the embedding with nodes corresponding to words and edges weighted by similarity.
  - To determine words with strong polarity, identify the words connected to multiple seed words through short but heavily weighted paths.

Load Data

- Download the 10-K and 10-Q financial reports data from Securities and Exchange Commission (SEC) via the Electronic Data Gathering, Analysis, and Retrieval (EDGAR) API using the **financeReports** helper function attached to this example as a supporting file. The financeReports function downloads 10-K and 10-Q reports for the specified year, quarter, and maximum character length.

Load Data

- Download a set of reports from the fourth quarter 2019 with fewer than 2 million characters.

```
year = 2019;

qtr = 4;

maxLength = 2e6;

textData = financeReports(year,qtr,maxLength);
```

Load Data

- Define sets of positive and negative seed words to use with this data.

```
seedsPositive = ["achieve" "advantage" "better" "creative" "efficiency" ...
 "efficiently" "enhance" "greater" "improved" "improving" ...
 "innovation" "innovations" "innovative" "opportunities" "profitable" ...
 "profitably" "strength" "strengthen" "strong" "success"]';
```

```
seedsNegative = ["adverse" "adversely" "against" "complaint" "concern" ...
 "damages" "default" "deficiencies" "disclosed" "failure" ...
 "fraud" "impairment" "litigation" "losses" "misleading" ...
 "omit" "restated" "restructuring" "termination" "weaknesses"]';
```

Prepare Text Data

- Create a function names preprocessText that prepares the text data for analysis. The preprocessText function, listed at the end of the example performs the following steps:
  - Erase any URLs.
  - Tokenize the text.
  - Remove tokens containing digits.
  - Convert the text to lower case.
  - Remove any words with two or fewer characters.
  - Remove any stop words.

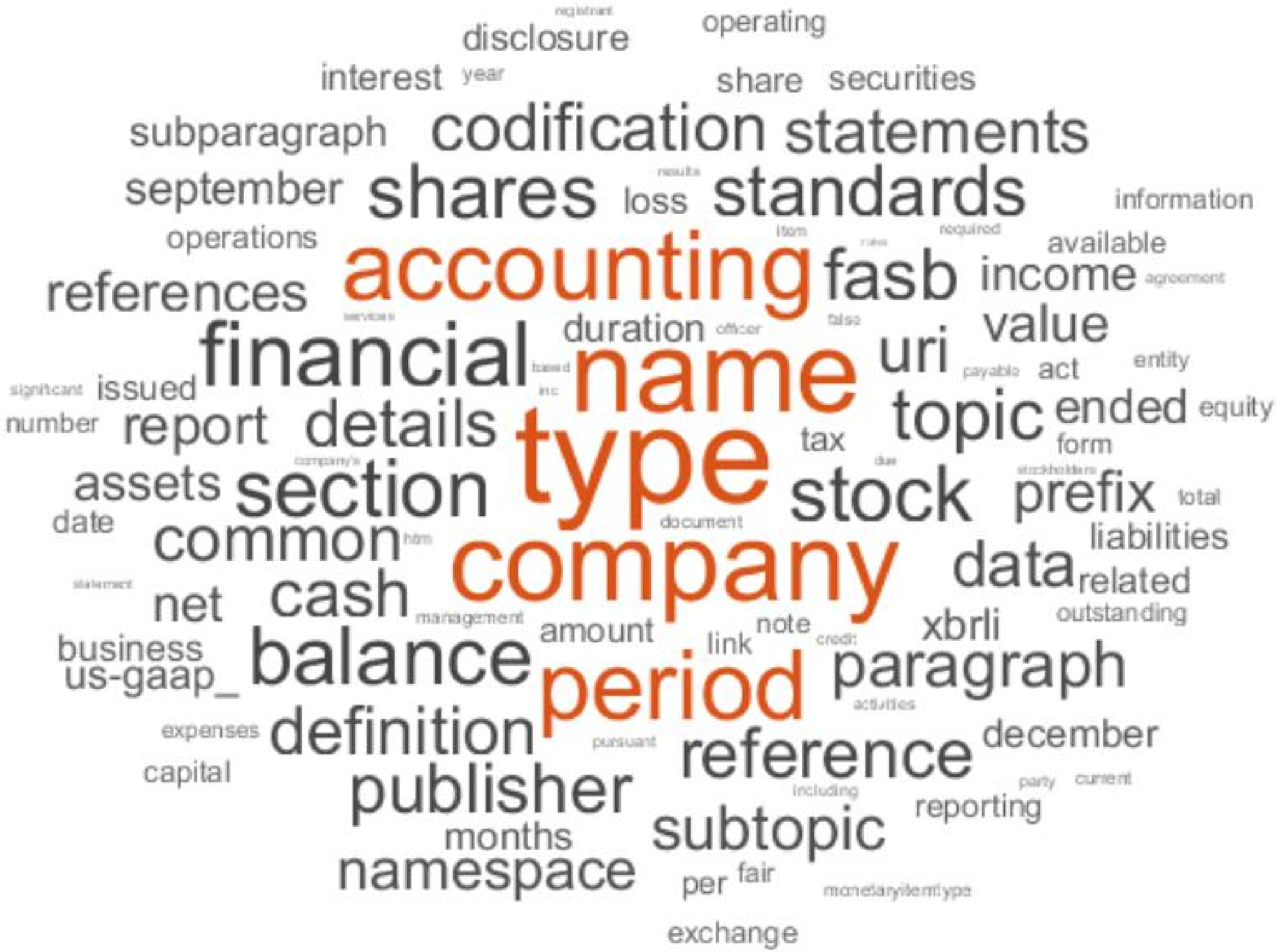
Prepare Text Data

- Preprocess the text using the preprocessText function. Depending on the size of the text data, this can take some time to run.

```
documents = preprocessText(textData);
```

- Visualize the preprocessed text data in a word cloud.

```
figure
wordcloud(documents);
```


registrant
operating
disclosure
interest
year
share
securities
subparagraph
codification
statements
september
shares
results
loss
standards
information
operations
item
rules
required
available
accounting
fasb
income
agreement
references
services
false
duration
officer
value
financial
name
uri
significant
issued
based
inc
payable
act
entity
number
report
details
topic
ended
equity
tax
form
company's
due
stockholders
assets
section
type
stock
prefix
total
date
document
liabilities
common
htm
data
related
statement
company
net
cash
management
outstanding
amount
note
credit
xbrli
business
link
us-gaap_
balance
period
paragraph
activities
expenses
definition
pursuant
december
reference
capital
including
party
current
publisher
reporting
months
subtopic
namespace
per
fair
monetaryitemtype
exchange

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.
- Train a word embedding that models the similarity between words using the training data. Specify a context window of size 25 and discard words that appear fewer than 20 times. Depending on the size of the text data, this can take some time to run.

Train Word Embedding

```
emb = trainWordEmbedding(documents,'Window',25,'MinCount',20);
```

```
Computing Vocabulary. Word count in millions: 12.
Vocabulary count: 17842.
Training: 0% Loss: 1.00369  Remaining time: 0 hours 47 minutes.
```

```
Computing Vocabulary. Word count in millions: 12.
Vocabulary count: 17842.
Training: 100% Loss: 0.325827 Remaining time: 0 hours 0 minutes.
```

Create Word Graph

- Create a weighted graph with nodes corresponding to words in the vocabulary, edges denoting whether the words are within a neigborhood of 7 of each other, and weights corresponding to the cosine distance between the corresponding word vectors in the embedding.

Create Word Graph

- For each word in the vocabulary, find the nearest 7 words and their cosine distances.

```
numNeighbors = 7;
vocabulary = emb.Vocabulary;
wordVectors = word2vec(emb,vocabulary);
[nearestWords,dist] = vec2word(emb,wordVectors,numNeighbors);
```

Create Word Graph

- To create the graph, use the graph function and specify pairwise source and target nodes, and specify their edge weights.
- Define the source and target nodes.

```
sourceNodes = repelem(vocabulary,numNeighbors);
targetNodes = reshape(nearestWords,1,[]);
```

- Calculate the edge weights.

```
edgeWeights = reshape(dist,1,[]);
```

Create Word Graph

- Create a graph connecting each word with its neigbors with edge weights corresponding to the similarity scores.

```
wordGraph = graph(sourceNodes,targetNodes,edgeWeights,vocabulary);
```

- Remove the repeated edges using the simplify function.

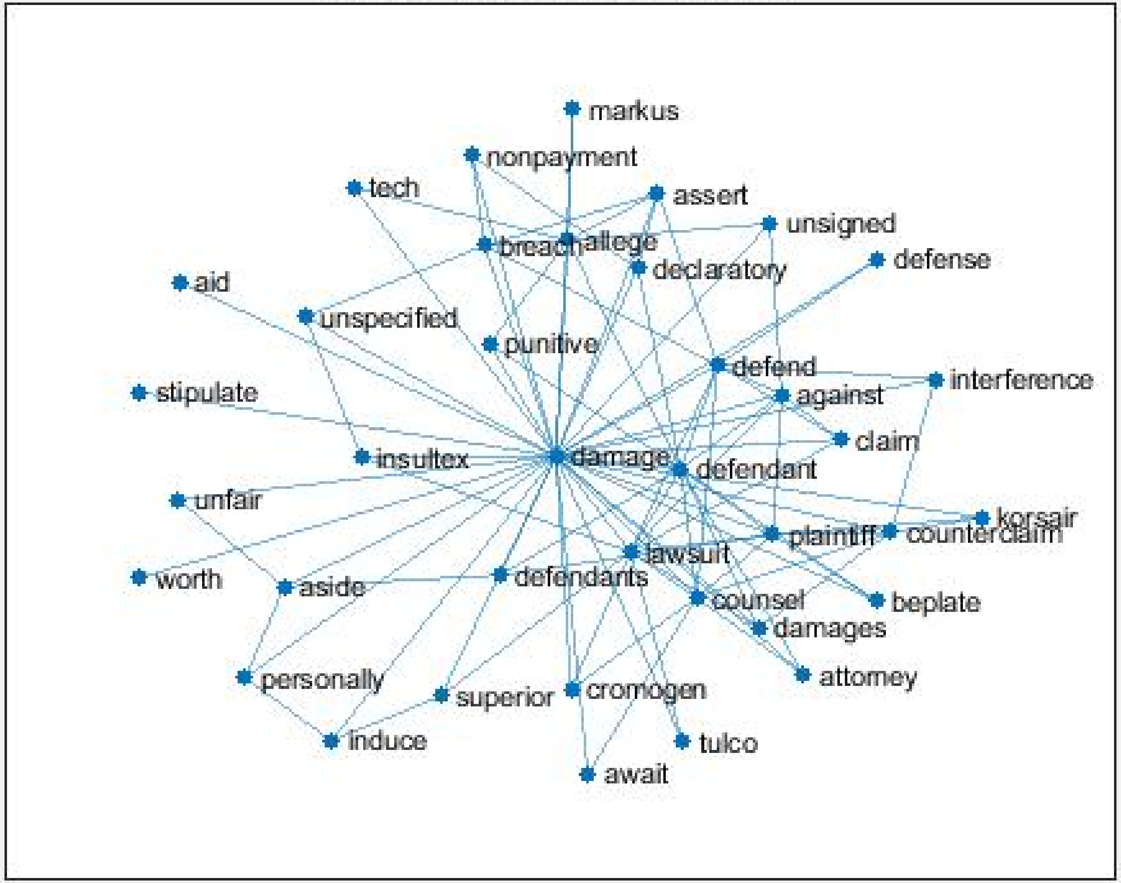
```
wordGraph = simplify(wordGraph);
```

Create Word Graph

- Visualize the section of the word graph connected to the word "damage".

```
word = "damage";
idx = findnode(wordGraph,word);
nbrs = neighbors(wordGraph,idx);
wordSubgraph = subgraph(wordGraph,[idx; nbrs]);
figure
plot(wordSubgraph)
title("Words connected to """ + word + """")
```

Words connected to "damage"



Generate Sentiment Scores

- To determine words with strong polarity, identify the words connected to multiple seed words through short but heavily weighted paths.
- Initialize an array of sentiment scores corresponding to each word in the vocabulary.

```
sentimentScores = zeros([1 numel(vocabulary)]);
```

Generate Sentiment Scores

- Iteratively traverse the graph and update the sentiment scores.
- Traverse the graph at different depths. For each depth, calculate the positive and negative polarity of the words by using the positive and negative seeds to propagate sentiment to the rest of the graph.
- For each depth:
  - Calculate the positive and negative polarity scores.
  - Account for the difference in overall mass of positive and negative flow in the graph.
  - For each node-word, normalize the difference of its two scores.

Generate Sentiment Scores

- After running the algorithm, if a phrase has a higher positive than negative polarity score, then its final polarity will be positive, and negative otherwise.
- Specify a maximum path length of 4.

```
maxPathLength = 4;
```

- Iteratively traverse the graph and calculate the sum of the sentiment scores.

Generate Sentiment Scores

```
for depth = 1:maxPathLength
    % Calculate polarity scores.
    polarityPositive = polarityScores(seedsPositive,vocabulary,wordGraph,depth);
    polarityNegative = polarityScores(seedsNegative,vocabulary,wordGraph,depth);
    % Account for difference in overall mass of positive and negative flow in the graph.
    b = sum(polarityPositive) / sum(polarityNegative);
    % Calculate new sentiment scores.
    sentimentScoresNew = polarityPositive - b * polarityNegative;
    sentimentScoresNew = normalize(sentimentScoresNew,'range',[-1,1]);
    % Add scores to sum.
    sentimentScores = sentimentScores + sentimentScoresNew;
end
```

Polarity Scores Function

- The polarityScores function returns a vector of polarity scores given a set of seed words, vocabulary, graph, and a specified depth. The function computes the sum over the maximum weighted path from every seed word to each node in the vocabulary. A high polarity score indicates phrases connected to multiple seed words via both short and strongly weighted paths.

Generate Sentiment Scores

- Normalize the sentiment scores by the number of iterations.

```
sentimentScores = sentimentScores / maxPathLength;
```

- Create a table containing the vocabulary and the corresponding sentiment scores.

```
tbl = table;
tbl.Token = vocabulary';
tbl.SentimentScore = sentimentScores';
```

Generate Sentiment Scores

- To remove tokens with neutral sentiment from the lexicon, remove the tokens with sentiment score that have absolute value less than a threshold of 0.1.

```
thr = 0.1;
idx = abs(tbl.SentimentScore) < thr;
tbl(idx,:) = [];
```

- Sort the table rows by descending sentiment score and view the first few rows.

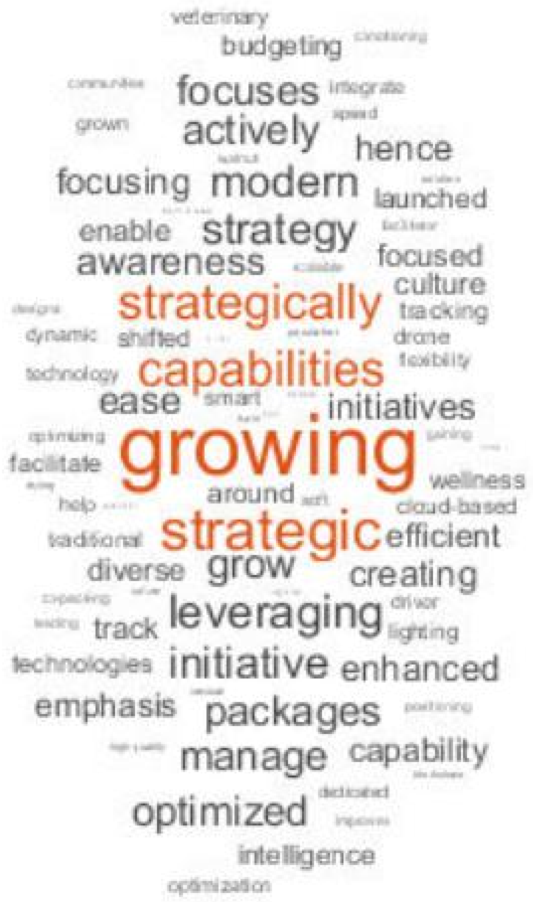
```
tbl = sortrows(tbl,'SentimentScore','descend');
```

Generate Sentiment Scores

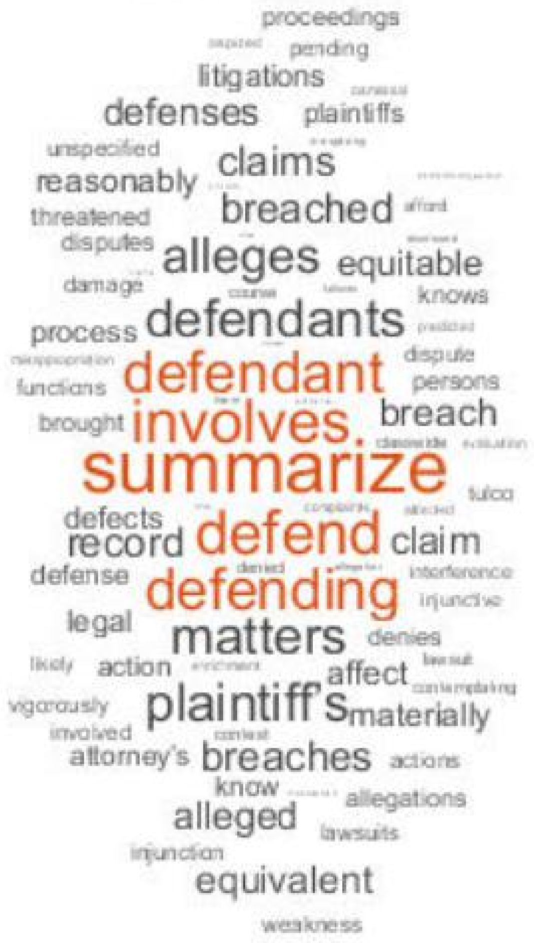
- You can use this table as a custom sentiment lexicon for the vaderSentimentScores function.
- Visualize the sentiment lexicon in word clouds. Display tokens with a positive score in one word cloud and tokens with negative scores in another. Display the words with sizes given by the absolute value their corresponding sentiment score.

Generate Sentiment Scores

```
figure
subplot(1,2,1);
idx = tbl.SentimentScore > 0;
tblPositive = tbl(idx,:);
wordcloud(tblTopPositive,'Token','SentimentScore')
title('Positive Words')
subplot(1,2,2);
idx = tbl.SentimentScore < 0;
tblNegative = tbl(idx,:);
tblNegative.SentimentScore = abs(tblNegative.SentimentScore);
wordcloud(tblTopNegative,'Token','SentimentScore')
title('Negative Words')
```



Positive Words



Negative Words

Generate Sentiment Scores

- Export the table to a CSV file.

```
filename = "financeSentimentLexicon.csv";
writetable(tbl,filename)
```

Analyze Sentiment in Text

- To analyze the sentiment in for previously unseen text data, preprocess the text using the same preprocessing steps and use the vaderSentimentScores function.
- Create a string array containing the text data and preprocess it using the preprocessText function.

```
textDataNew = [
 "This company is showing extremely strong growth."
 "This other company is accused of misleading consumers."];
documentsNew = preprocessText(textDataNew);
```

Analyze Sentiment in Text

- Evaluate the sentiment using the vaderSentimentScores function. Specify the sentiment lexicon created in this example using the 'SentimentLexicon' option.

```
compoundScores = vaderSentimentScores(documentsNew,'SentimentLexicon',tbl)
compoundScores = 2×1
 0.2834
 -0.1273
```

Train a Sentiment Classifier

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

Train a Sentiment Classifier

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

Load Pretrained Word Embedding

- Word embeddings map words in a vocabulary to numeric vectors. These embeddings can capture semantic details of the words so that similar words have similar vectors. They also model relationships between words through vector arithmetic. For example, the relationship Rome is to Paris as Italy is to France is described by the equation Rome – Italy + France = Paris. Load a pretrained word embedding using the fastTextWordEmbedding function.

```
emb = fastTextWordEmbedding;
```

Load Opinion Lexicon

- Load the positive and negative words from the opinion lexicon (also known as a sentiment lexicon) from https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html. First, extract the files from the .rar file into a folder named opinion-lexicon-English, and then import the text.
- Load the data using the function readLexicon listed at the end of this example. The output data is a table with variables Word containing the words, and Label containing a categorical sentiment label, Positive or Negative.

```
data = readLexicon;
```

Load Opinion Lexicon

- View the first few words labeled as positive.

```
idx = data.Label == "Positive";
head(data(idx,:))
ans=8×2 table
 Word Label

 _____________ _________
 "a+" Positive
 "abound" Positive
 "abounds" Positive
 "abundance" Positive
 "abundant" Positive ......
```

Load Opinion Lexicon

- View the first few words labeled as negative.

```
idx = data.Label == "Negative";
head(data(idx,:))
ans=8×2 table
 Word Label

 _______________ _________
 "2-faced" Negative
 "2-faces" Negative
 "abnormal" Negative
 "abolish" Negative
 "abominable" Negative......
```

Prepare Data for Training

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

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

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

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

Prepare Data for Training

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

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

Train Sentiment Classifier

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

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

Test Classifier

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

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

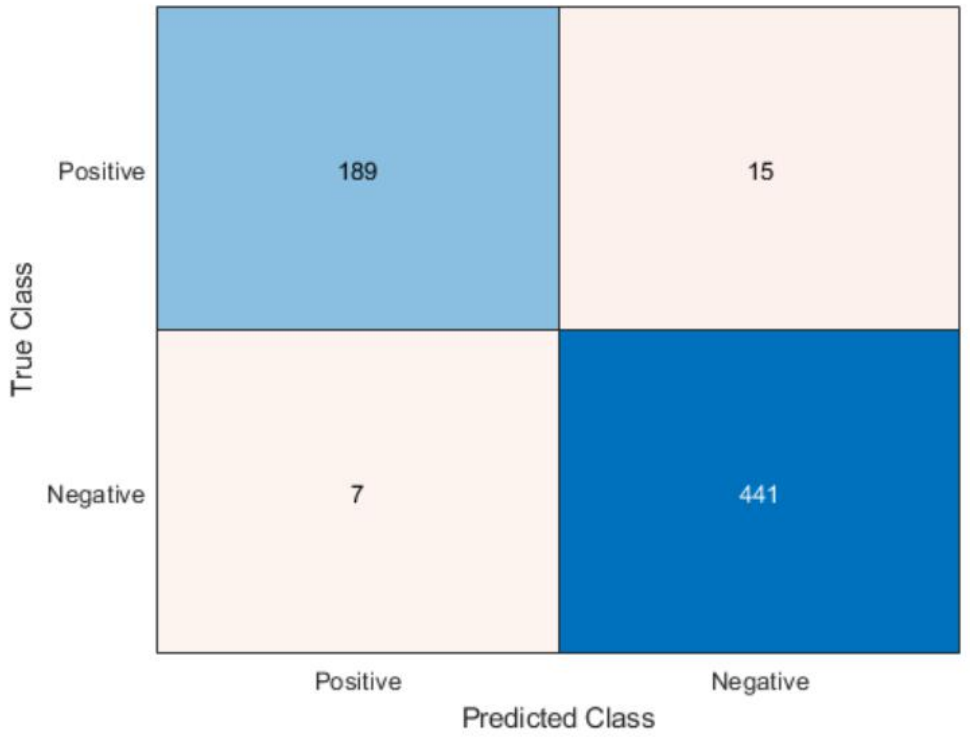
- Predict the sentiment labels of the test word vectors.

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

Test Classifier

- Visualize the classification accuracy in a confusion matrix.

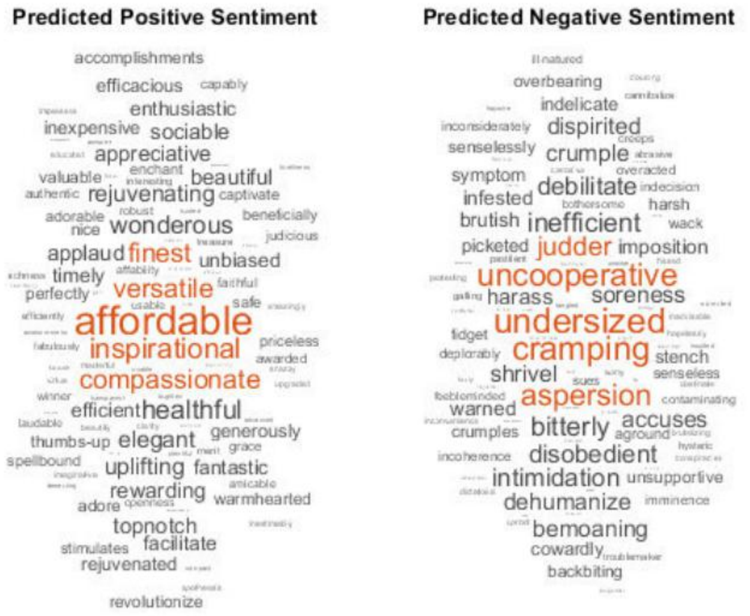
```
figure
confusionchart(YTest,YPred);
```



Test Classifier

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

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



Calculate Sentiment of Collections of Text

- To calculate the sentiment of a piece of text, for example an update on social media, predict the sentiment score of each word in the text and take the mean sentiment score.

```
filename = "weekendUpdates.xlsx";
tbl = readtable(filename,'TextType','string');
textData = tbl.TextData;
textData(1:3)
ans = 3×1 string array
 "Happy anniversary! ❤ Next stop: Paris! ✈ #vacation"
 "Haha, BBQ on the beach, engage smug mode!   ❤  #vacation"
 "getting ready for Saturday night  #yum #weekend "
```

Calculate Sentiment of Collections of Text

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

```
documents = preprocessText(textData);
```

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

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

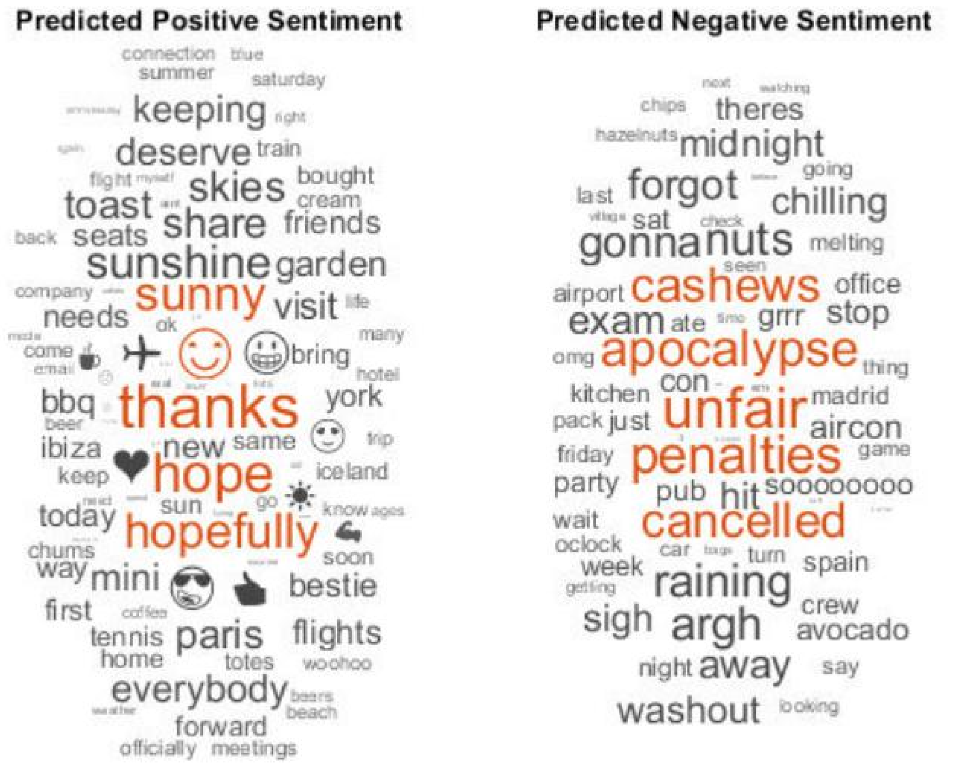
Calculate Sentiment of Collections of Text

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

```
words = documents.Vocabulary;
words(ismember(words,wordsTrain)) = [];
vec = word2vec(emb,words);
[YPred,scores] = predict(mdl,vec);
```

Calculate Sentiment of Collections of Text

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



Calculate Sentiment of Collections of Text

- To calculate the sentiment of a given piece of text, compute the sentiment score for each word in the text and calculate the mean sentiment score.
- Calculate the mean sentiment score of the updates. For each document, convert the words to word vectors, predict the sentiment score on the word vectors, transform the scores using the score-to-posterior transform function and then calculate the mean sentiment score.

Calculate Sentiment of Collections of Text

```
for i = 1:numel(documents)
   words = string(documents(i));
   vec = word2vec(emb,words);
   [~,scores] = predict(mdl,vec);
   sentimentScore(i) = mean(scores(:,1));
end
```

Calculate Sentiment of Collections of Text

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

```
table(sentimentScore', textData)
```

```
ans=50×2 table

 Var1 textData

 1.8382 "Happy anniversary! ❤ Next stop: Paris! ✈ #vacation"

 1.294 "Haha, BBQ on the beach, engage smug mode!   ❤  #vacation"

 1.0922 "getting ready for Saturday night  #yum #weekend
```