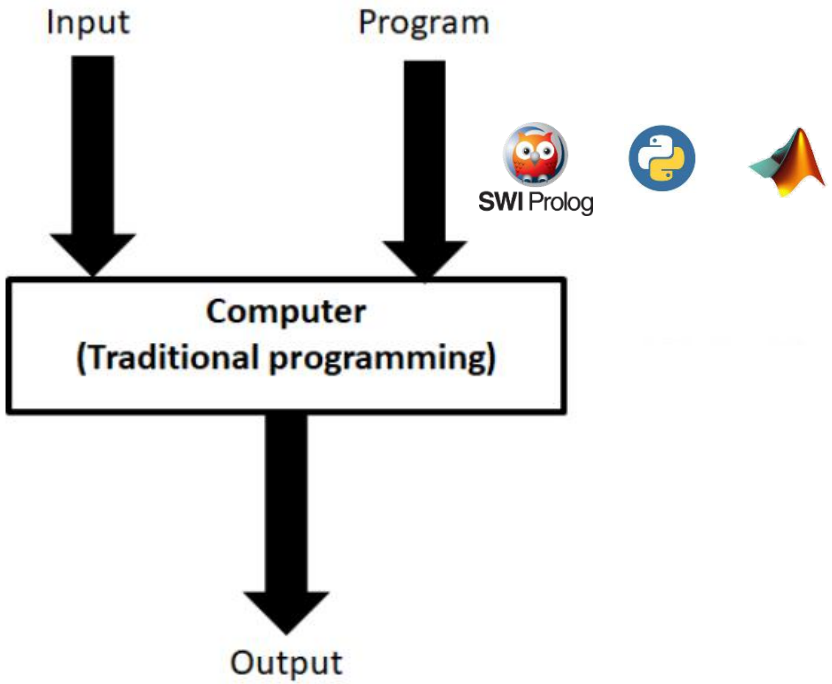
Language Models

- for Text Analysis

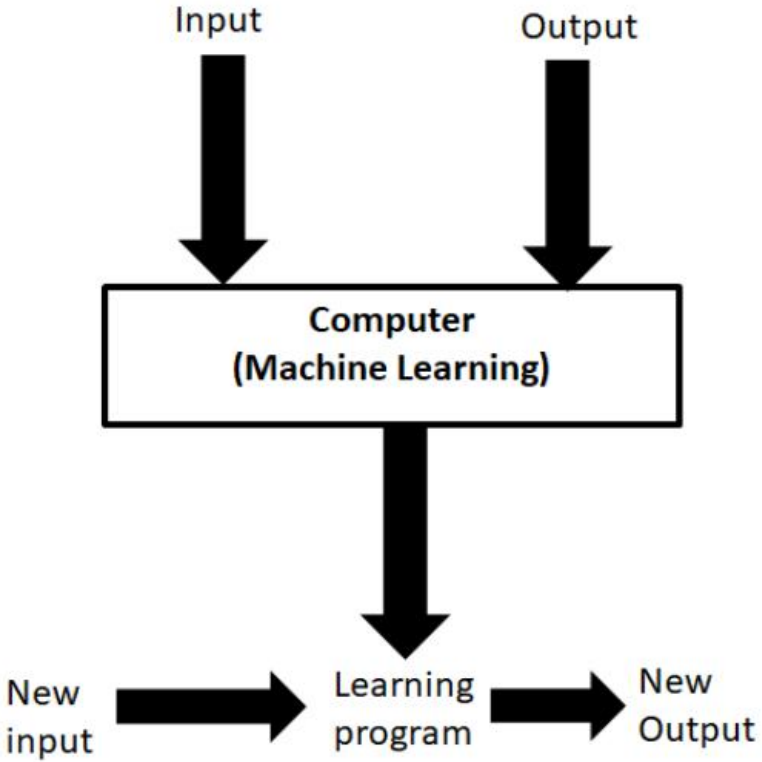
Outline

- Machine Learning vs GOFAI (Symbolic artificial intelligence)
- About Text Analysis
- Text Analytics in MATLAB
- Import and Visualize Text Data
- Preprocess Text Data
- Bag-of-words
- TFIDF

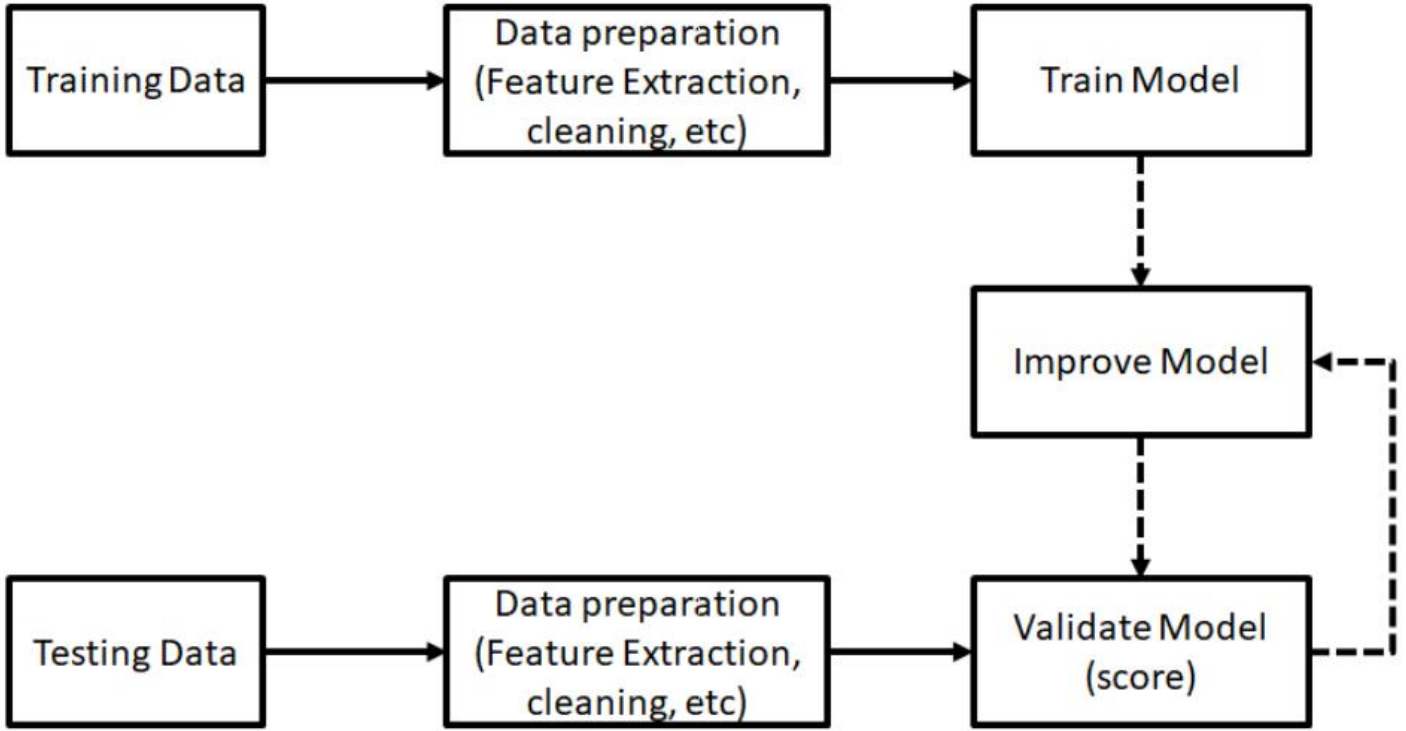
GOFAI


Input
Program
SWI Prolog
Computer
(Traditional programming)
Output

Machine Learning



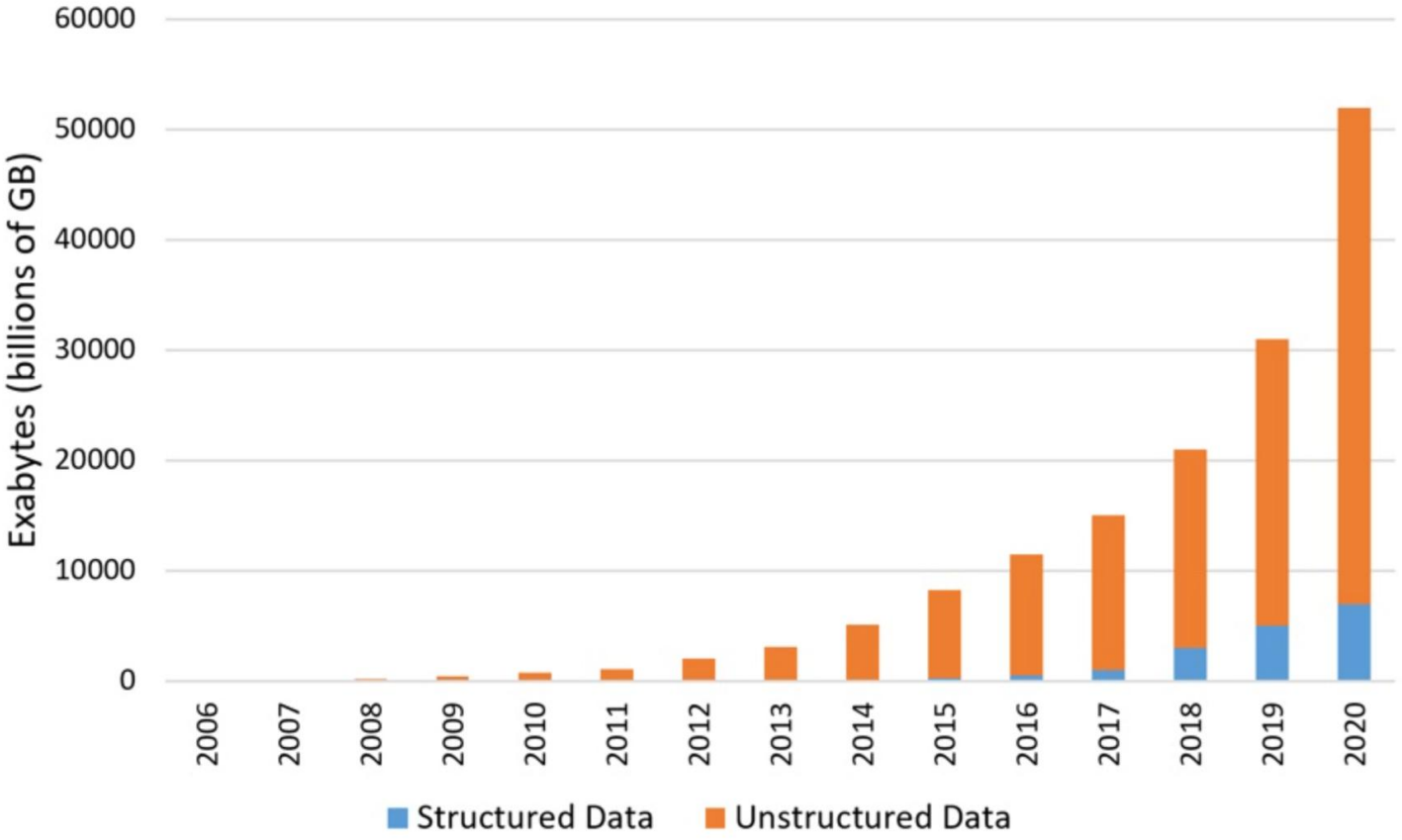
Machine Learning



About Text Analysis

- Textual data also has huge business value, and companies can use this data to help profile customers and understand customer trends. This can either be used to offer a more personalized experience for users or as information for targeted marketing.

The Cambrian Explosion…of Data



About Text Analysis

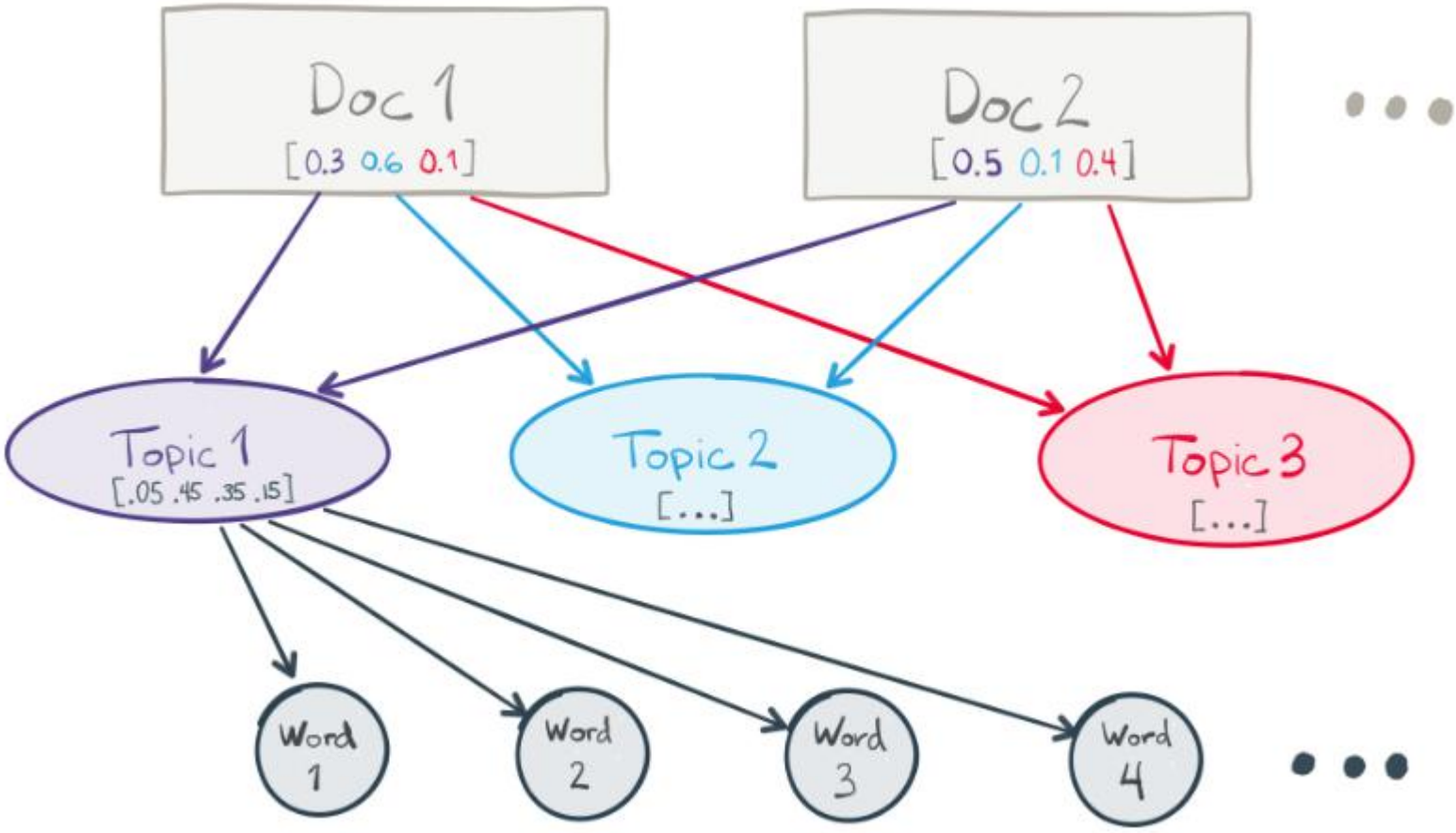
- Text analysis can be understood as the technique of gleaning useful information from text. This can be done through various techniques, and we use **Natural Language Processing** (**NLP**), **Computational Linguistics** (**CL**), and numerical tools to get this information.

About Text Analysis

- Natural language processing (NLP) refers to the use of a computer to process natural language. For example, removing all occurrences of the word *thereby* from a body of text is one such example, albeit a basic example.
- Computational linguistics (CL), as the name suggests, is the study of linguistics from a computational perspective. This means using computers and algorithms to perform linguistics tasks such as marking your text as a part of speech (such as noun or verb), instead of performing this task manually.

About Text Analysis

- Information retrieval (IR) builds on statistical approaches in text processing and allows us to classify, cluster, and retrieve documents. Methods such as topic modeling can help us identify key topics in large, unstructured bodies of text. Identifying these topics goes beyond searching for keywords, and we use statistical models to further understand the underlying nature of bodies of text.


Doc 1
[0.3 0.6 0.1]
Doc 2
[0.5 0.1 0.4]
Topic 1
[.05 .45 .35 .15]
Topic 2
[...]
Topic 3
[...]
Word 1
Word 2
Word 3
Word 4

Text Analytics in MATLAB

- Text Analytics Toolbox provides algorithms and visualizations for preprocessing, analyzing, and modeling text data. Models created with the toolbox can be used in applications such as sentiment analysis, predictive maintenance, and topic modeling.

Text Analytics in MATLAB

- Text Analytics Toolbox includes tools for processing raw text from sources such as equipment logs, news feeds, surveys, operator reports, and social media. You can extract text from popular file formats, preprocess raw text, extract individual words, convert text into numerical representations, and build statistical models.

Text Analytics in MATLAB

- Using machine learning techniques such as LSA, LDA, and word embeddings, you can find clusters and create features from high-dimensional text datasets. Features created with Text Analytics Toolbox can be combined with features from other data sources to build machine learning models that take advantage of textual, numeric, and other types of data.

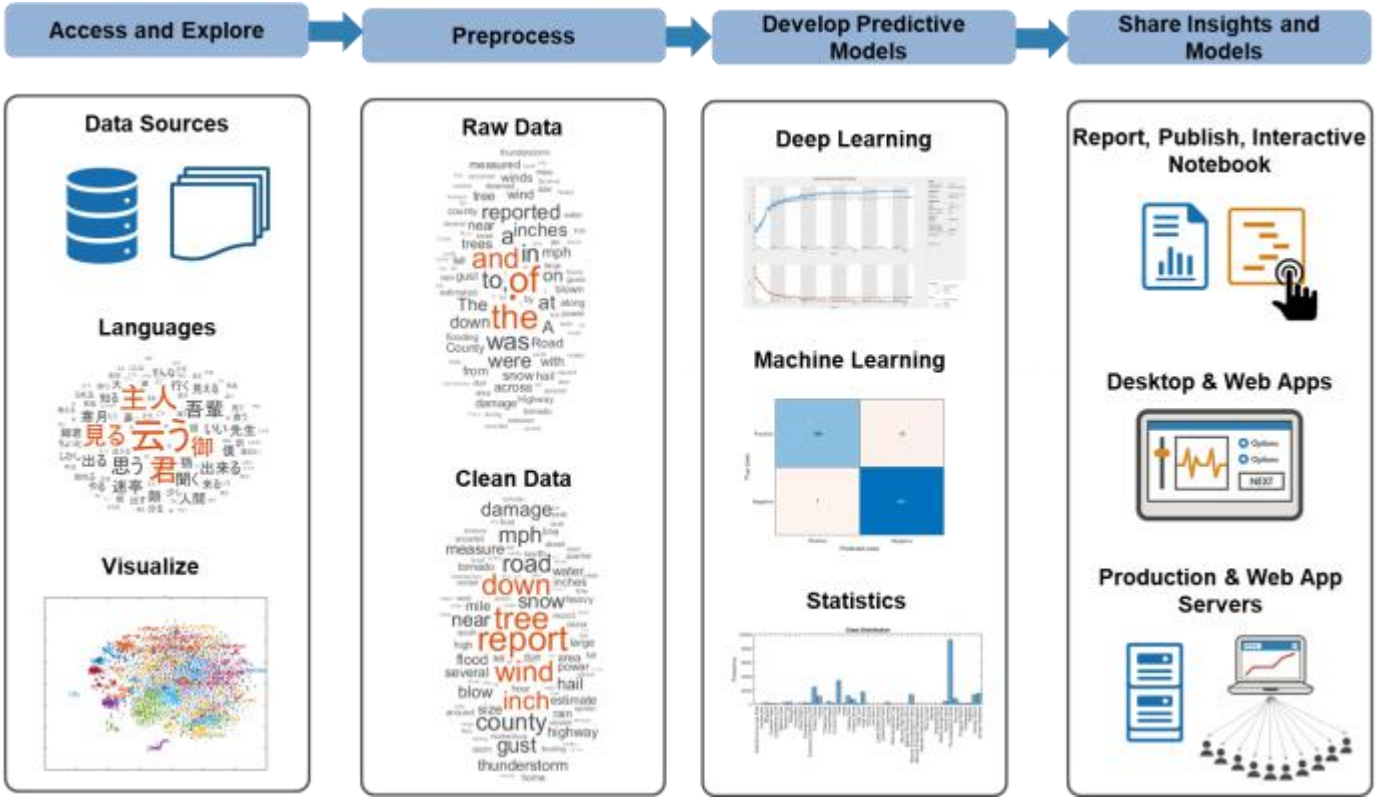
Text Analytics in MATLAB

Text Analytics Workflow

An end-to-end text analytics workflow involves the following four steps:

- Access data from databases, the web, and internal file repositories and explore by visualization.
- Preprocess data by eliminating extraneous information such as punctuation, common words, or stop words such as “a” and “the.”
- Build predictive models by using machine or deep learning algorithms.
- Share insights and use predictive models in applications.

Text Analytics in MATLAB



Import and Visualize Text Data

Extract Text Data from Files

- We will show how to extract the text data from text, HTML, Microsoft Word, PDF, CSV, and Microsoft Excel files and import it into MATLAB for analysis.
- Usually, the easiest way to import text data into MATLAB is to use the **extractFileText** function. This function extracts the text data from text, PDF, HTML, and Microsoft Word files. To import text from CSV and Microsoft Excel files, use **readtable**. To extract text from HTML code, use **extractHTMLText**. To read data from PDF forms, use **readPDFFormData**.

Text File

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

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

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

```
start = " I" + newline;
fin = " II";
sonnet1 = extractBetween(str,start,fin)
sonnet1 =
   "
      From fairest creatures we desire increase,
      That thereby beauty's rose might never die,
      But as the riper should by time decease, ……
```

Text File

- For text files containing multiple documents seperated by newline characters, use the readlines function.

filename = "multilineSonnets.txt";

str = readlines(filename)

str = 3×1 string

"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."

"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, ……

Microsoft Word Document

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

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

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

```
start = " II" + newline;
fin = " III";
sonnet2 = extractBetween(str,start,fin)
sonnet2 =
   " When forty winters shall besiege thy brow,

     And dig deep trenches in thy beauty's field,

     Thy youth's proud livery so gazed on now,     ……
```

Microsoft Word Document

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

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

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days; ……

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;  ……
```

PDF Files

- 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 Files

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

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

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

```
start = newline + "IV" + newline;
fin = newline + "V" + newline;
sonnet4 = extractBetween(str,start,fin)
sonnet4 =
    "
     Unthrifty loveliness, why dost thou spend
     Upon thy self thy beauty's legacy?
     Nature's bequest gives nothing, but doth lend, ……
```

HTML Files

- 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"
```

HTML Files

- 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

     Analyze and model text data

     Release Notes

     PDF Documentation

     Release Notes ……
```

Parse HTML Code

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

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

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

```
subtrees(1:10)
ans =
  10×1 htmlTree:
    <A class="skip_link sr-only" href="#content_container">Skip to content</A>
    <A href="https://www.mathworks.com?s_tid=gn_logo" class="svg_link navbar-brand"><IMG src="/images/responsive/global/pic-header-mathworks-logo.svg" class="mw_logo" alt="MathWorks"/></A> ……
```

Parse HTML Code

str = extractHTMLText(subtrees);

- View the extracted text of the first 10 hyperlinks.

str(1:10)

ans = 10×1 string

"Skip to content"

""

"Products"

"Solutions"

"Academia"

"Support"

"Community"

"Events"

"Get MATLAB"

""

Parse HTML Code

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

```
attr = "href";
str = getAttribute(subtrees(1:10),attr)
str = 10×1 string
   "#content_container"
   "https://www.mathworks.com?s_tid=gn_logo"
   "https://www.mathworks.com/products.html?s_tid=gn_ps"
   "https://www.mathworks.com/solutions.html?s_tid=gn_sol"
   "https://www.mathworks.com/academia.html?s_tid=gn_acad"
   "https://www.mathworks.com/support.html?s_tid=gn_supp"
   "https://www.mathworks.com/matlabcentral/?s_tid=gn_mlc"
   "https://www.mathworks.com/company/events.html?s_tid=gn_ev"
   "https://www.mathworks.com/products/get-matlab.html?s_tid=gn_getml"
   "https://www.mathworks.com?s_tid=gn_logo"
```

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 table data from factoryReposts.csv using the readtable function and view the first few rows of the table.

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

ans=8×5 table

Description	Category	Urgency	Resolution	Cost
"Items are occasionally getting stuck in the scanner spools."	"Mechanical Failure"	"Medium"	"Readjust Machine"	45
"Loud rattling and banging sounds are coming from assembler pistons."	"Mechanical Failure"	"Medium"	"Readjust Machine"	35
"There are cuts to the power when starting the plant."	"Electronic Failure"	"High"	"Full Replacement"	16200
"Fried capacitors in the assembler."	"Electronic Failure"	"High"	"Replace Components"	352
"Mixer tripped the fuses."	"Electronic Failure"	"Low"	"Add to Watch List"	55
"Burst pipe in the constructing agent is spraying coolant."	"Leak"	"High"	"Replace Components"	371
"A fuse is blown in the mixer."	"Electronic Failure"	"Low"	"Replace Components"	441
"Things continue to tumble off of the belt."	"Mechanical Failure"	"Low"	"Readjust Machine"	38

CSV and Microsoft Excel Files

- Extract the text data from the event_narrative column and view the first few strings.

```
str = T.Description;
str(1:10)
ans = 10×1 string
    "Items are occasionally getting stuck in the scanner spools."
    "Loud rattling and banging sounds are coming from assembler pistons."
    "There are cuts to the power when starting the plant."
    "Fried capacitors in the assembler."
    "Mixer tripped the fuses."
    "Burst pipe in the constructing agent is spraying coolant."
    "A fuse is blown in the mixer."
    "Things continue to tumble off of the belt."
    "Falling items from the conveyor belt."
    "The scanner reel is split, it will soon begin to curve."
```

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

```
location = fullfile(matlabroot,"examples","textanalytics","data","exampleSonnet*.txt");
fds = fileDatastore(location,'ReadFcn',@extractFileText)
fds =
  FileDatastore with properties:
                Files: {
                    ' ...\matlab\examples\textanalytics\data\exampleSonnet1.txt';
                    ' ...\matlab\examples\textanalytics\data\exampleSonnet2.txt';
                    ' ...\matlab\examples\textanalytics\data\exampleSonnet3.txt'
                     ... and 2 more
                    }
              Folders: {
                    ' ...\matlab\examples\textanalytics\data'
                    } ……
```

Extract Text from Multiple Files

- 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 = 5×1 string

“ 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." ……

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 factoryReports.csv contains factory reports, including a text description and categorical labels for each event.

```
filename = "factoryReports.csv";
tbl = readtable(filename,'TextType','string');
```

Extract the text data from the Description column.

```
textData = tbl.Description;
textData(1:10)
ans = 10x1 string
   "Items are occasionally getting stuck in the scanner spools."
   "Loud rattling and banging sounds are coming from assembler pistons."
   "There are cuts to the power when starting the plant."  ……
```

Visualize Text Data Using Word Clouds

- Create a word cloud from the reports.

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

Visualize Text Data Using Word Clouds

- Compare the words in the reports with labels "Leak" and "Mechanical Failure". 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 = tbl.Category;

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

subplot(1,2,2)
idx = labels == "Mechanical Failure";
wordcloud(textData(idx),'Color','magenta');
title("Mechanical Failure")
```

Visualize Text Data Using Word Clouds

- Compare the words in the reports with urgency "Low", "Medium", and "High".

```
figure
urgency = tbl.Urgency;
subplot(1,3,1)
idx = urgency == "Low";
wordcloud(textData(idx));
title("Urgency: Low")
subplot(1,3,2)
idx = urgency == "Medium";
wordcloud(textData(idx));
title("Urgency: Medium")
subplot(1,3,3)
idx = urgency == "High";
wordcloud(textData(idx));
title("Urgency: High")
```

Visualize Text Data Using Word Clouds

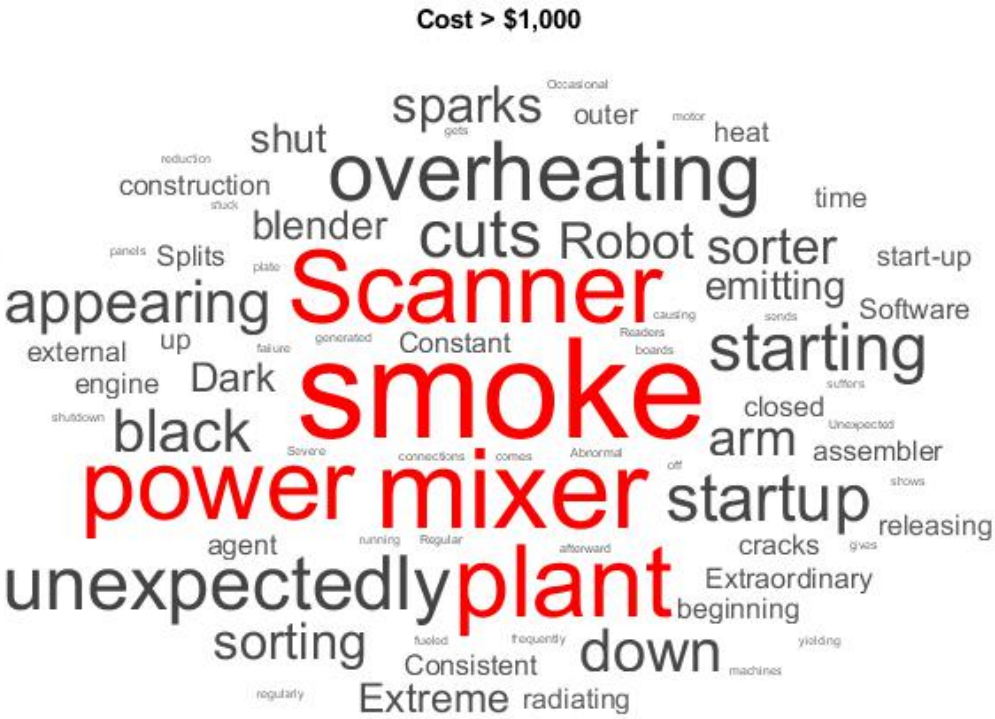
- Compare the words in the reports with cost reported in hundreds of dollars to the reports with costs reported in thousands of dollars. Create word clouds of the reports for each of these amounts with highlight color blue and red respectively.

```
cost = tbl.Cost;
idx = cost > 100;
figure
wordcloud(textData(idx),'HighlightColor','blue');
title("Cost > $100")
```

Visualize Text Data Using Word Clouds

- Compare the words in the reports with cost reported in hundreds of dollars to the reports with costs reported in thousands of dollars. Create word clouds of the reports for each of these amounts with highlight color blue and red respectively.

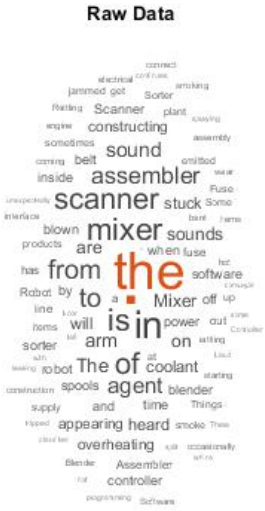
```
idx = cost > 1000;
figure
wordcloud(textData(idx),'HighlightColor','red');
title("Cost > $1,000")
```



Preprocess Text Data

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



Preprocess Text Data

- Load the example data. The file factoryReports.csv contains factory reports, including a text description and categorical labels for each event.

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

- Extract the text data from the field Description, and the label data from the field Category.

```
textData = data.Description;
labels = data.Category;
textData(1:10)
ans = 10×1 string
    "Items are occasionally getting stuck in the scanner spools."
    "Loud rattling and banging sounds are coming from assembler pistons."
    "There are cuts to the power when starting the plant."
    "Fried capacitors in the assembler."
    "Mixer tripped the fuses."
    “Burst pipe in the constructing agent is spraying coolant." ……
```

Preprocess Text Data

- Create an array of tokenized documents.

```
cleanedDocuments = tokenizedDocument(textData);
cleanedDocuments(1:10)
ans =
  10×1 tokenizedDocument:

    10 tokens: Items are occasionally getting stuck in the scanner spools .
    11 tokens: Loud rattling and banging sounds are coming from assembler pistons .
    11 tokens: There are cuts to the power when starting the plant .
     6 tokens: Fried capacitors in the assembler .
     5 tokens: Mixer tripped the fuses .
    10 tokens: Burst pipe in the constructing agent is spraying coolant .
     8 tokens: A fuse is blown in the mixer .
     9 tokens: Things continue to tumble off of the belt .
     7 tokens: Falling items from the conveyor belt . ……
```

Preprocess Text Data

- To improve lemmatization, add part of speech details to the documents using addPartOfSpeechDetails. Use the addPartOfSpeech function before removing stop words and lemmatizing.

```
cleanedDocuments = addPartOfSpeechDetails(cleanedDocuments);
```

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

```
cleanedDocuments = removeStopWords(cleanedDocuments);
cleanedDocuments(1:10)
ans =
  10×1 tokenizedDocument:

    7 tokens: Items occasionally getting stuck scanner spools .
    8 tokens: Loud rattling banging sounds coming assembler pistons .
    5 tokens: cuts power starting plant . ……
```

Preprocess Text Data

- Lemmatize the words using normalizeWords.

```
cleanedDocuments = normalizeWords(cleanedDocuments,'Style','lemma');
cleanedDocuments(1:10)
ans =
  10×1 tokenizedDocument:

    7 tokens: items occasionally get stuck scanner spool .
    8 tokens: loud rattle bang sound come assembler piston .
    5 tokens: cut power start plant .
    4 tokens: fry capacitor assembler .
    4 tokens: mixer trip fuse .
    7 tokens: burst pipe constructing agent spray coolant .
    4 tokens: fuse blow mixer .
    6 tokens: thing continue tumble off belt .
    5 tokens: fall item conveyor belt . ……
```

Preprocess Text Data

- Erase the punctuation from the documents.

```
cleanedDocuments = erasePunctuation(cleanedDocuments);
cleanedDocuments(1:10)
ans =
  10×1 tokenizedDocument:

    6 tokens: items occasionally get stuck scanner spool
    7 tokens: loud rattle bang sound come assembler piston
    4 tokens: cut power start plant
    3 tokens: fry capacitor assembler
    3 tokens: mixer trip fuse
    6 tokens: burst pipe constructing agent spray coolant
    3 tokens: fuse blow mixer
    5 tokens: thing continue tumble off belt
    4 tokens: fall item conveyor belt  ……
```

Preprocess Text Data

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

```
cleanedDocuments = removeShortWords(cleanedDocuments,2);
cleanedDocuments = removeLongWords(cleanedDocuments,15);
cleanedDocuments(1:10)
ans =
  10×1 tokenizedDocument:

    6 tokens: items occasionally get stuck scanner spool
    7 tokens: loud rattle bang sound come assembler piston
    4 tokens: cut power start plant
    3 tokens: fry capacitor assembler
    3 tokens: mixer trip fuse
    6 tokens: burst pipe constructing agent spray coolant
    3 tokens: fuse blow mixer
    5 tokens: thing continue tumble off belt ……
```

Preprocess Text Data

- Create a bag-of-words model.

```
cleanedBag = bagOfWords(cleanedDocuments)
cleanedBag =
  bagOfWords with properties:

         Counts: [480×352 double]
     Vocabulary: [1×352 string]
       NumWords: 352
   NumDocuments: 480
```

Preprocess Text Data

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

```
cleanedBag = removeInfrequentWords(cleanedBag,2)
cleanedBag =
  bagOfWords with properties:

         Counts: [480×163 double]
     Vocabulary: [1×163 string]
       NumWords: 163
   NumDocuments: 480
```

Preprocess Text Data

- 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.

```
[cleanedBag,idx] = removeEmptyDocuments(cleanedBag);
labels(idx) = [];
cleanedBag
cleanedBag =
  bagOfWords with properties:

         Counts: [480×163 double]
     Vocabulary: [1×163 string]
       NumWords: 163
   NumDocuments: 480
```

Preprocess Text Data

- Compare the preprocessed data with the raw data.

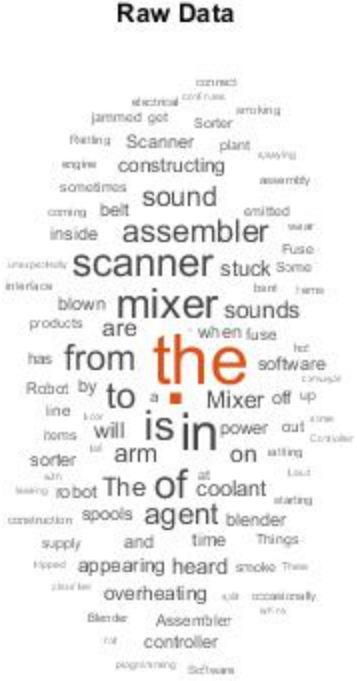
```
rawDocuments = tokenizedDocument(textData);
rawBag = bagOfWords(rawDocuments)
rawBag =
  bagOfWords with properties:

         Counts: [480×555 double]
     Vocabulary: [1×555 string]
       NumWords: 555
   NumDocuments: 480
```

Preprocess Text Data

- 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(cleanedBag);
title("Cleaned Data")
```



Bag-of-words

- The bag-of-words model is arguably the most straightforward form of representing a sentence as a vector. Let's start with an example:

S1:"The dog sat by the mat."

S2:"The cat loves the dog."

- If we follow some preprocessing steps, we will end up with the following sentences:

S1:"dog sat mat."

S2:"cat love dog."

Bag-of-words

- These will now look like this:

S1:['dog', 'sat', 'mat']

S2:['cat', 'love', 'dog']

- If we want to represent this as a vector, we would need to first construct our vocabulary, which would be the unique words found in the sentences. Our vocabulary vector is now as follows:

Vocab = ['dog', 'sat', 'mat', 'love', 'cat']

Bag-of-words

- This means that our representation of our sentences will also be vectors with a length of 5 - we can also say that our vectors will have 5 dimensions. We can also think of mapping of each word in our vocabulary to a number (or index), in which case we can also refer to our vocabulary as a dictionary.
- The bag-of-words model involves using word frequencies to construct our vectors. What will our sentences now look like?

S1:[1, 1, 1, 0, 0]

S2:[1, 0, 0, 1, 1]

Bag-of-words

- It's easy enough to understand - there is 1 occurrence of dog, the first word in the vocabulary, and 0 occurrences of love in the first sentence, so the appropriate indexes are given the value based on the word frequency. If the first sentence has 2 occurrences of the word dog, it would be represented as:

S1: [2, 1, 1, 0, 0]

Bag-of-words

- One important feature of the bag-of-words model which we must remember is that it is an order less document representation - only the counts of the words matter. We can see that in our example above as well, where by looking at the resulting sentence vectors we do not know which words came first. This leads to a loss in spatial information, and by extension, semantic information. However, in a lot of information retrieval algorithms, the order of the words is not important, and just the occurrences of the words are enough for us to start with.

Bag-of-words

- An example where the bag of words model can be used is in spam filtering - emails that are marked as spam are likely to contain spam-related words, such as buy, money, and stock. By converting the text in emails into a bag of words models, we can use Bayesian probability to determine if it is more likely for a mail to be in the spam folder or not.

TF-IDF

- TF-IDF is short for term frequency-inverse document frequency. Largely used in searchengines to find relevant documents based on a query, it is a rather intuitive approach to converting our sentences into vectors.
- TF-IDF tries to encode two different kinds of information - term frequency and inverse document frequency. Term frequency (TF) is the number of times a word appears in a document.

TF-IDF

- IDF helps us understand the importance of a word in a document. By calculating the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term) and then taking the logarithm of that quotient, we can have a measure of how common or rare the word is among all documents.

TF-IDF

- In case the preceding explanation wasn't very clear, expressing them as formulas will help!

TF(t) = (number of times term t appears in a document) / (total number of terms in the document)

IDF(t) = log_e (total number of documents / number of documents with term t in it)

TF-IDF

- TF-IDF is simply the product of these two factors - TF and IDF. Together it encapsulates more information into the vector representation, instead of just using the count of the words like in the bag-of-words vector representation. TF-IDF makes rare words more prominent and ignores common words such as is, of, and that, which may appear a lot of times, but have little importance.

TF-IDF

- Create a Term Frequency–Inverse Document Frequency (tf-idf) matrix from a bag-of-words model.
- Load the example data. The file sonnetsPreprocessed.txt contains preprocessed versions of Shakespeare's sonnets. The file contains one sonnet per line, with words separated by a space. Extract the text from sonnetsPreprocessed.txt, split the text into documents at newline characters, and then tokenize the documents.

```
filename = "sonnetsPreprocessed.txt";
str = extractFileText(filename);
textData = split(str,newline);
documents = tokenizedDocument(textData);
```

TF-IDF

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

```
bag = bagOfWords(documents)
bag =
  bagOfWords with properties:

         Counts: [154x3092 double]
     Vocabulary: ["fairest"    "creatures"    "desire"    ...    ]
       NumWords: 3092
   NumDocuments: 154
```

TF-IDF

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

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

ans = 10×10

3.6507	4.3438	2.7344	3.6507	4.3438	2.2644	3.2452	3.8918	2.4720	2.5520
0	0	0	0	0	4.5287	0	0	0	0
0	0	0	0	0	0	0	0	2.552	0
0	0	0	0	0	2.2644	0	0	0	0
0	0	0	0	0	2.2644	0	0	0	0
0	0	0	0	0	2.2644	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2.2644	0	0	0	2.5520
0	0	2.7344	0	0	0	0	0	0	0