TU 257 – Fundamentals of Data Science

Data Analytics

L11 – Text Mining

Brendan Tierney

Agenda

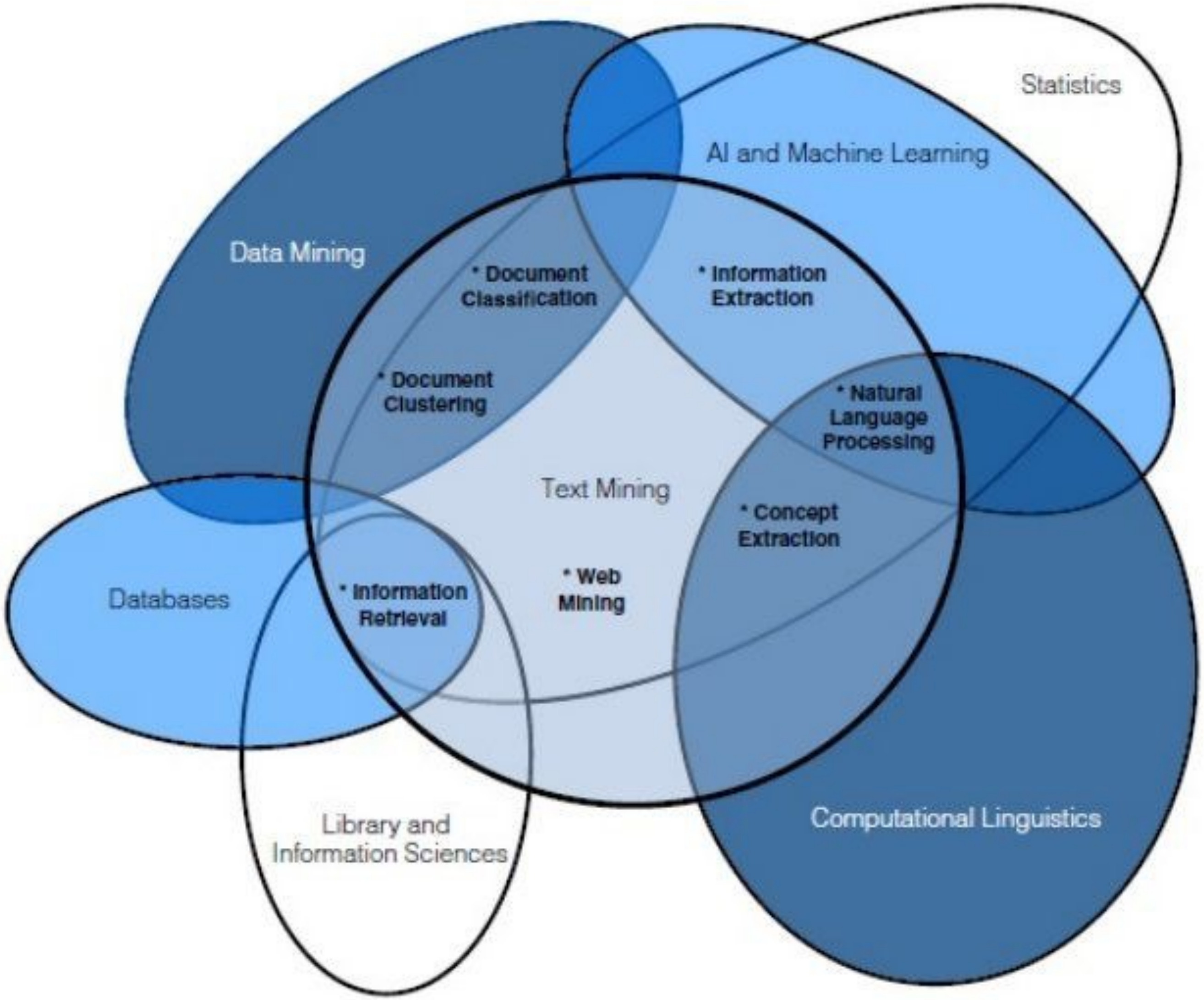
- Introduction to Text Mining
- Application Areas for Text Mining
- Key Terms
- (Basics of) Text Mining Process
- Demo - Walk through of example
- Term Frequency Inverse Document Frequency
- N-grams
- Typical Application Areas in the Enterprise

Text Mining

- Data is available in different formats
- Most common is Structured Data ! (Semi-structured data)
  - We know the structure and data formats
  - Most common type as it is captured by most applications and processes
- Less Common, but has been increasing is Unstructured Data
  - Data that has no structure
  - Typical Examples include
    - Text
    - Audio
    - Images
    - Videos
    - etc.)
- In most scenarios when people talk about Unstructured Data they actually talking about (Semi-) Structured Data
  - JSON objects

Text Mining

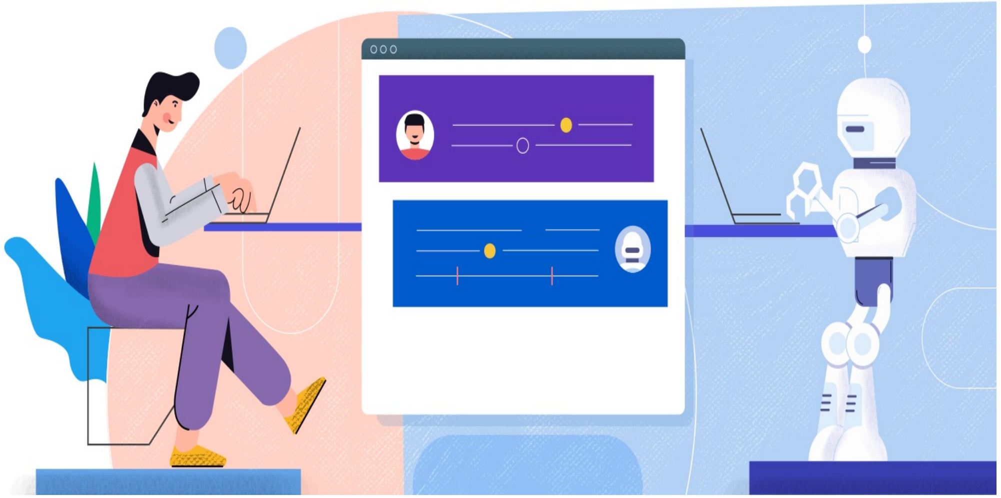
- Typical ML / DM / DS, work with a cleaned and organised table of rows and columns fed as input to the algorithm.
- Unlike numeric or categorical data, natural language does not exist in a structured format consisting of rows and columns.
- Fundamental step involves converting text into semi-structured data.
- Once in semi-structured format, can apply a variety of visualisations, mathematical, statistical and machine learning algorithms to gain insights and patterns.


Statistics
AI and Machine Learning
Data Mining
* Document Classification
* Information Extraction
* Document Clustering
* Natural Language Processing
Text Mining
* Concept Extraction
Databases
* Information Retrieval
* Web Mining
Library and Information Sciences
Computational Linguistics

Different types of Text Mining

- Search and Information Retrieval (IR): Storage and retrieval of text documents, including search engines and keyword search.
- Document Clustering: Grouping and categorizing terms, snippets, paragraphs or documents using data mining clustering methods.
- Document Classification: Grouping and categorizing snippets, paragraphs, or document using data mining classification methods, based on models trained on labelled examples.
- Web Mining: Data and Text Mining on the Internet with a specific focus on the scale and interconnectedness of the web.
- Information Extraction (IE): Identification and extraction of relevant facts and relationships from unstructured text; the process of making structured data from unstructured and semi-structured text.
- Natural Language Processing (NLP): Low-level language processing and understanding tasks (e.g., tagging part of speech); often used synonymously with computational linguistics
- Concept Extraction: Grouping of words and phrases into semantically similar group.

Other applications areas



Chatbots

Search Engines

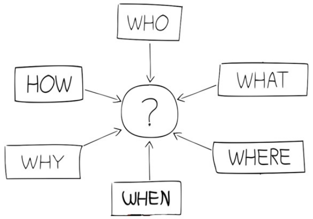
Some Basics

- Corpus
  - A collection of documents is called a corpus
  - A document consists of a set of tokens.
  - A token is a contiguous string of characters that does not contain a separator.
  - A separator is a special character such as a blank or mark of punctuation.
  - A term is a token with a specific meaning in a given language.
- AKA – our Data (set)

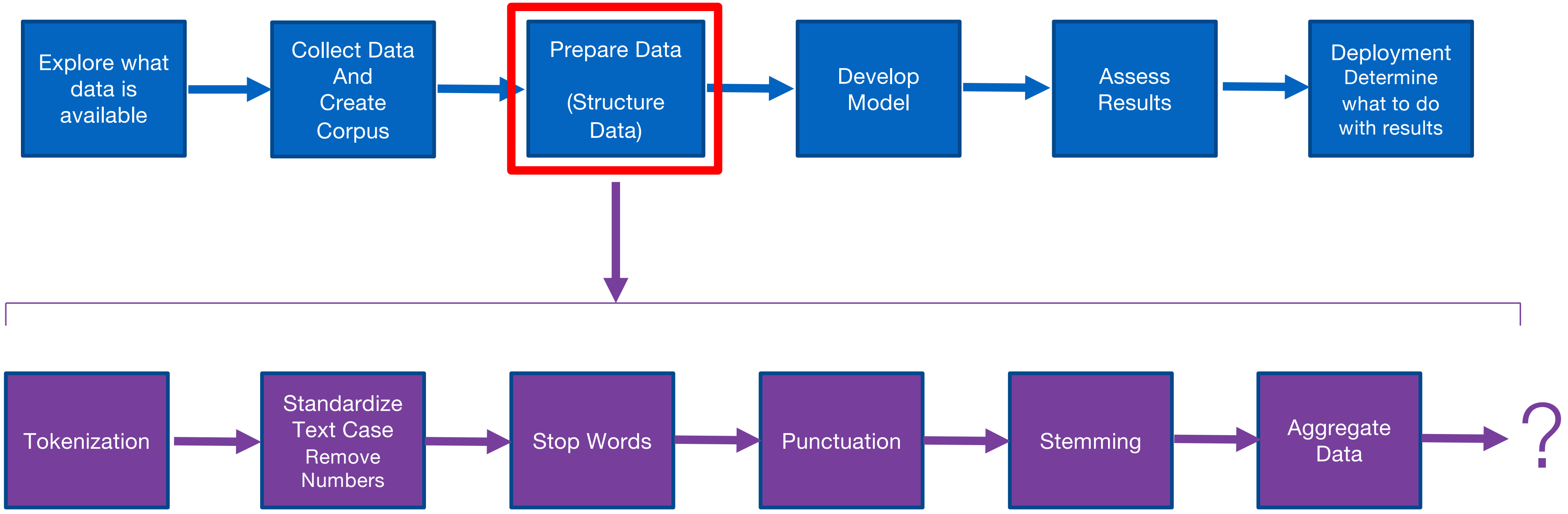
  My name is John. I live in New York Zip 90210. My dog loves to eat pizza all day.

  - Web pages
  - PDF documents
  - Word documents
  - PPT
  - etc

Processing Text – Core Steps



Determine the objective / business question

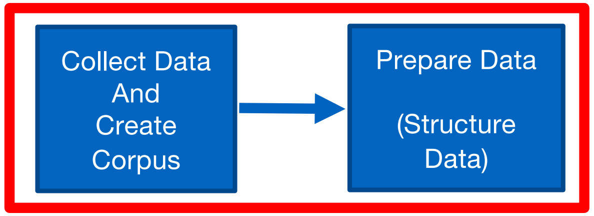


Explore what data is available → Collect Data And Create Corpus → Prepare Data (Structure Data) → Develop Model → Assess Results → Deployment Determine what to do with results

Tokenization → Standardize Text Case Remove Numbers → Stop Words → Punctuation → Stemming → Aggregate Data → ?

Very similar to CRISP-DM process

Step 1 – Load and Prepare Data

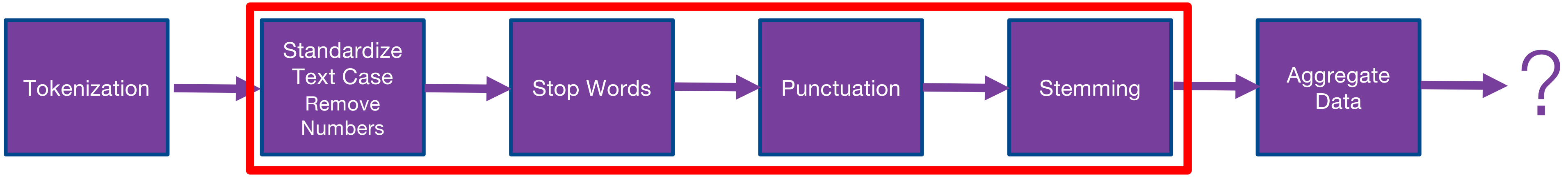


- Every tool and language has libraries to do all the hard work
- Need to work out the best ones to use – these are constantly evolving and new libraries are being created

```
import io import pdfminer
from pprint import pprint
from pdfminer.converter import TextConverter
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfpage import PDFPage

#directory were manifestos are located wkDir =
'.../General_Election_Ire/’

#define the names of the Manifesto PDF files & setup party flag
pdfFile = wkDir+'FGManifesto16_2.pdf’
party = 'FG’

#pdfFile = wkDir+'Fianna_Fail_GE_2016.pdf’
#party = 'FF’
#pdfFile = wkDir+'Labour_GE_2016.pdf’
#party = 'LB’
#pdfFile = wkDir+'Sinn_Fein_GE_2016.pdf’
#party = 'SF'
```

```
install.packages (c ( "tm", "wordcloud", "RCurl", "XML", "SnowballC"))

# load 'tm' packages
library (tm)
library (wordcloud)
library (SnowballC)

#load htmlToText
source("/Users/brendan.tierney/htmltotext.R")

data1 <- htmlToText(" http://oralytics.com ")
data2 <- htmlToText(" http://www.rte.ie/news ")
data3 <- htmlToText(" http://www.tudublin.ie")

# merge/combine the data
data <- c(data1, data2)
data <- c(data, data3)
data <- c(data, data4)
```

Extra work is needed to tidy up the text, remove unusual characters, tags, etc



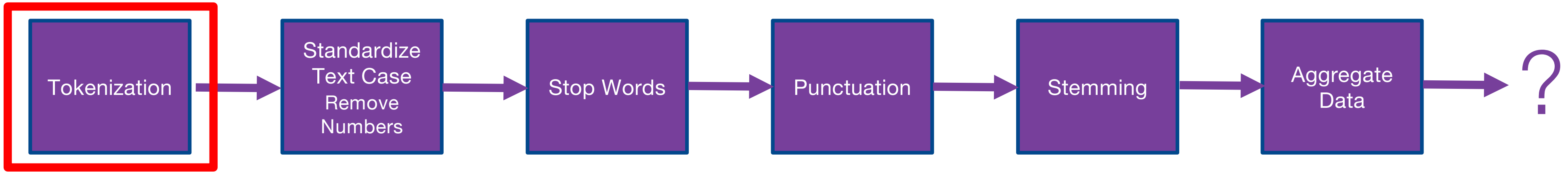
The order of some of these steps might need to be swapped

No Free Lunch – You will need to experiment to see what works.

It Depends! - Can depend on domain, context, what words are used/common, etc.

What you do at each steps remains the same, it’s just the order.

- Although some changes might be needed based on It Depends! (see above)

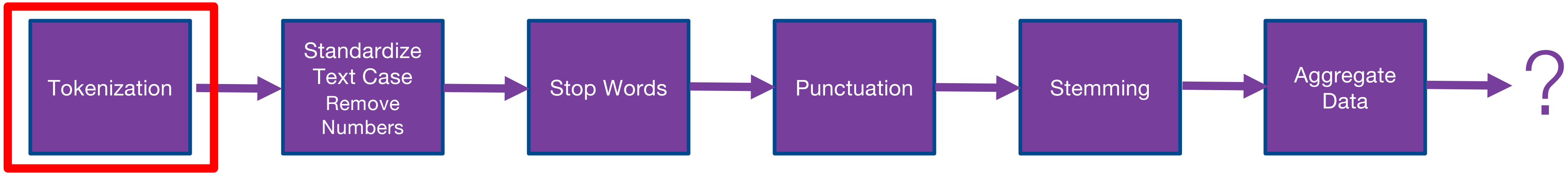


Tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens.

- The list of tokens becomes input for further processing such as parsing or text mining

- All contiguous strings of alphabetic characters are part of one token; likewise with numbers.
- Tokens are separated by whitespace characters, such as a space or line break, or by punctuation characters.
- Punctuation and whitespace may or may not be included in the resulting list of tokens.

- Each word is called a token and the process of discretising words within document called tokenisation.
- For now, a document is simply a collection of tokens (bag of words).



Tokenization

Standardize Text Case Remove Numbers

Stop Words

Punctuation

Stemming

Aggregate Data

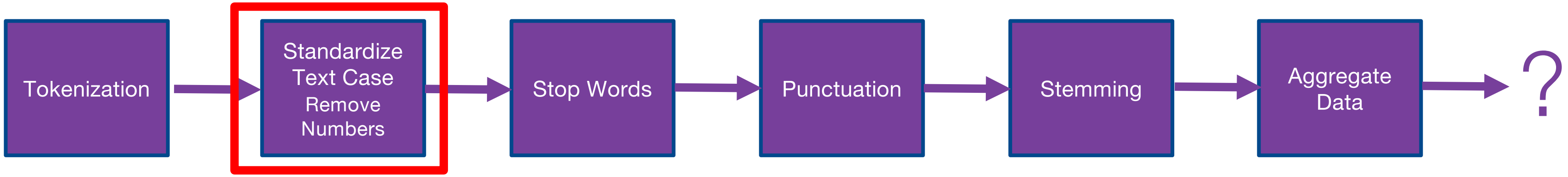
?

My name is John. I live in New York Zip 90210. My dog loves to eat pizza all day.

My	name	is	John.	I	live	in	New	York	Zip	90210.	My	dog	loves	to	eat	Pizza	all	Day.

```
#tokenise the data set
from nltk.tokenize import sent_tokenize, word_tokenize

words = word_tokenize(text)

print(words)
```

```
# create a corpus
txt_corpus <- Corpus (VectorSource (data))
```



Having mixed case can cause Duplicate words

Pizza = pizza ?

New = new ?

Brendan = brendan ?

Convert to all upper or lower case

- *Lower case is typical*

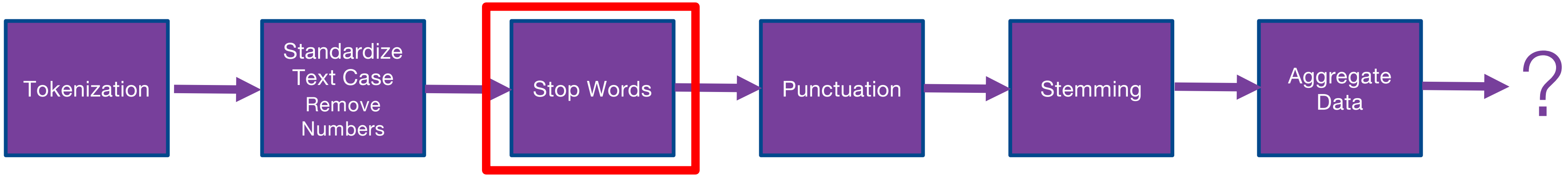
Numbers have No (?) meaning – They can be removed.

My name is John. I live in New York Zip 90210. My dog loves to eat pizza all day.

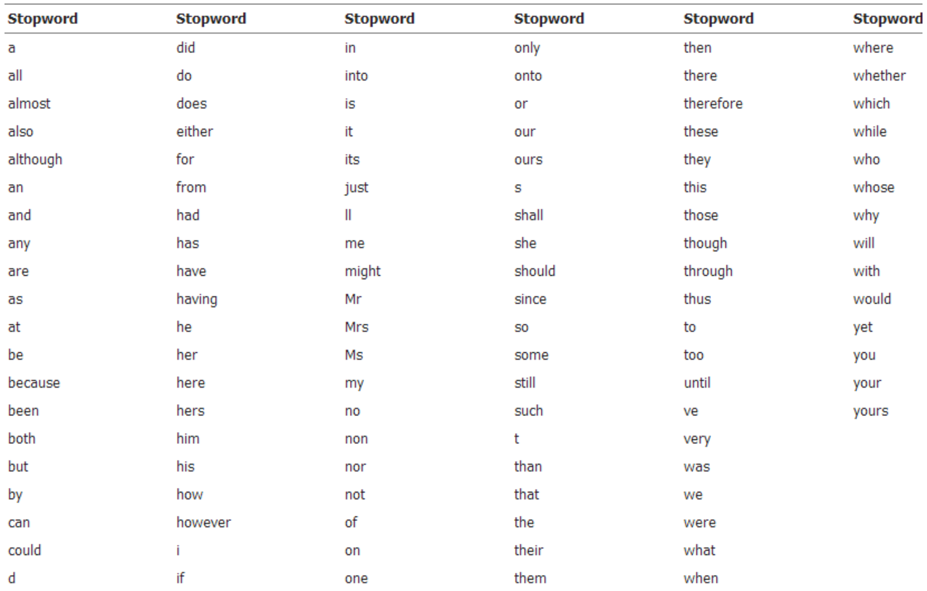
my	name	is	john.	i	live	in	new	york	zip	90210.	my	dog	loves	to	eat	pizza	all	day.

```
#converts to lower case, removes numbers and punctuation
wordsFiltered = [tokens.lower() for tokens in tokens if tokens.isalpha()]

print(len(wordsFiltered)) print(wordsFiltered)
```

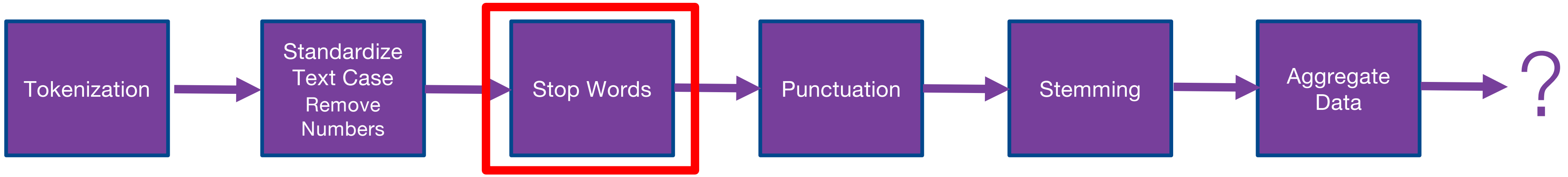
my	name	is	john	i	live	in	new	york	zip	~~90210.~~	my	dog	loves	to	eat	pizza	all	day



- Common words such as “a”, “this”, “and” and other similar terms occur in documents.
- Clearly, in larger documents, a higher number of such terms can be expected
- They do not convey specific meaning.
- Most grammatical necessities e.g. articles, conjunctions, prepositions, and pronouns may be filtered before additional analysis is performed.
- Such terms are called Stop Words.
- Most languages and tools come with a standard set of Stop Words for each language.
  - You can add extra Stop Words which are domain/context dependent.



Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
a	did	in	only	then	where
all	do	into	onto	there	whether
almost	does	is	or	therefore	which
also	either	it	our	these	while
although	for	its	ours	they	who
an	from	just	s	this	whose
and	had	ll	shall	those	why
any	has	me	she	though	will
are	have	might	should	through	with
as	having	Mr	since	thus	would
at	he	Mrs	so	to	yet
be	her	Ms	some	too	you
because	here	my	still	until	your
been	hers	no	such	ve	yours
both	him	non	t	very	
but	his	nor	than	was	
by	how	not	that	we	
can	however	of	the	were	
could	i	on	their	what	
d	if	one	them	when	

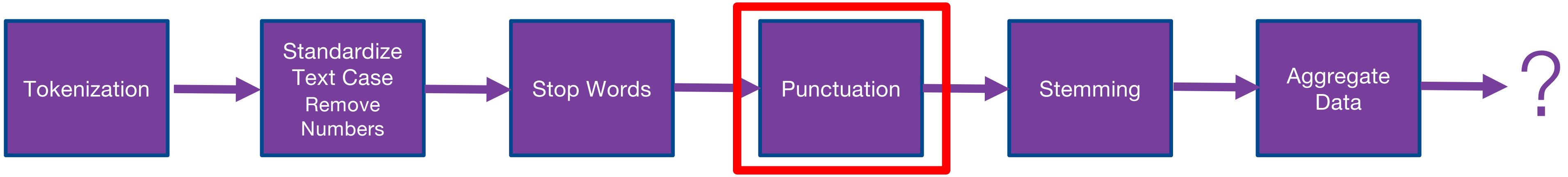


My name is John. I live in New York Zip 90210. My dog loves to eat pizza all day.

my name is john i live in new york zip my dog loves to eat pizza all day

name john live new york zip dog loves eat pizza day

```
#We initialize the stopwords variable which is a list of words like
#"The", "I", "and", etc. that don't hold much value as keywords
stop_words = stopwords.words('english’)
print(stop_words)

#some extra stop words are needed after examining the data and word cloud
#these are added
extra_stop_words = ['ireland','irish','need', 'also', 'set', 'within', 'use', 'order', 'would', 'year’]
stop_words.extend(extra_stop_words)
print(stop_words)

# remove stop words from tokenised data set
filtered_words = [word for word in wordsFiltered if word not in stop_words]
print(filtered_words)
```



- Characters that are defined as punctuations are removed from a token before text indexing

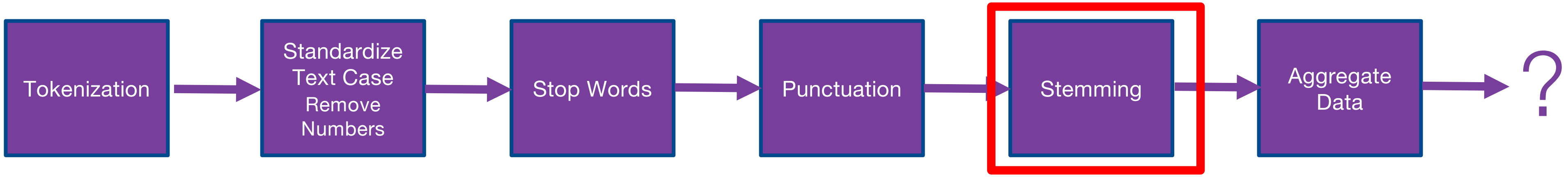
. , : ; ‘ @ ~ # { } [ ] + = - _ ( ) * & ^ % $ £ € “ ! ` ¬ ¦ \ |/ ?

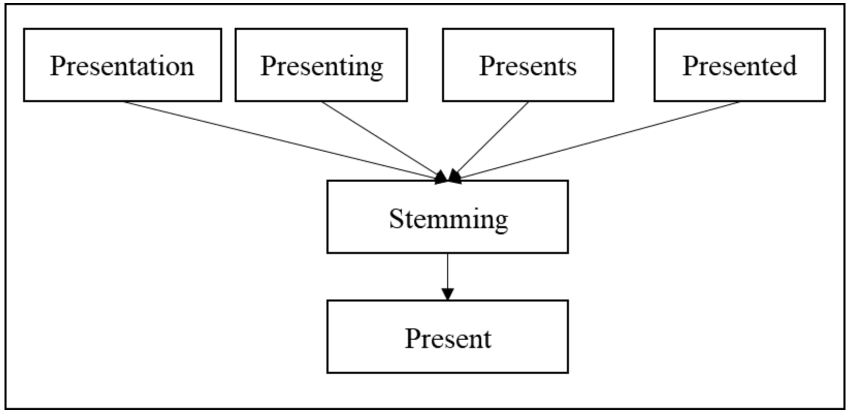
- Our examples have already dealt with this!

```
from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer(r'\w+')
tokenizer.tokenize('Eighty-seven miles to go, yet. Onward!')
```

- Other approaches
  - Create a list and remove
  - Use regular expressions
  - Use other libraries

```
import string

s = '... some string with punctuation ...’
s = s.translate(None, string.punctuation)
```





- Stemming reduces a word to its stem

- The result is less readable by humans but makes the test more comparable across observations

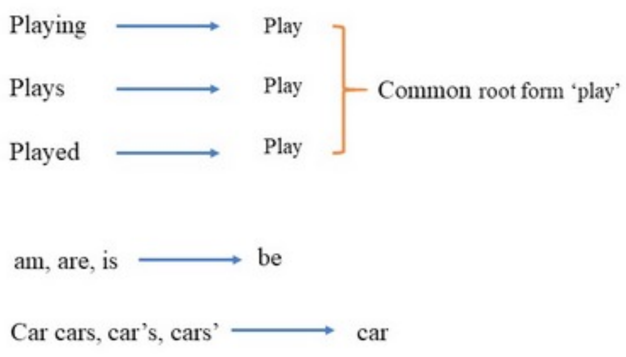
- Gets root/stem of the word

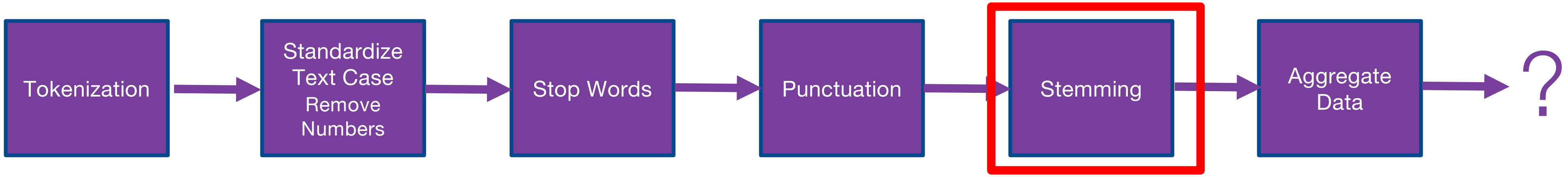
- Reduces the number of tokens in the final data set

- Needs careful exploration

- Porter stemming works according to rules where idea is to remove and/or replace suffix of words. Example rules include:
  - Replace terms which end in ‘ies’ by ‘y,’ e.g. “anomalies” with “anomaly.”
  - Stem all terms ending in “s” by removing the “s,” e.g. “algorithms” to “algorithm.”

- Porter stemmer is extremely efficient but can be error prone e.g. “arms” and “army” both stemmed to “arm,”





```
from nltk.stem import PorterStemmer

porter = PorterStemmer()

#provide a word to be stemmed
print("Porter Stemmer")                        Porter Stemmer
print(porter.stem("cats"))                     cat
print(porter.stem("trouble"))                  troubl
print(porter.stem("troubling"))                troubl
print(porter.stem("troubled"))                 troubl
```

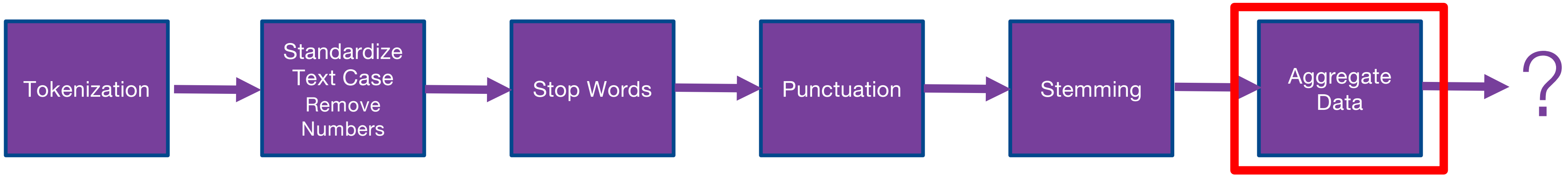
Other libraries available for non-English languages e.g. SnowballStemmers, ISRIStemmer, RSLPSStemmer

```
from nltk.stem.snowball import SnowballStemmer

englishStemmer=SnowballStemmer("english")
englishStemmer.stem("having")
```

https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

SnowballStemmers

- Danish
- Dutch
- English
- French
- German
- Hungarian
- Italian
- Norwegian
- Porter
- Portuguese
- Romanian
- Russian
- Spanish
- Swedish



Tokenization

Standardize Text Case Remove Numbers

Stop Words

Punctuation

Stemming

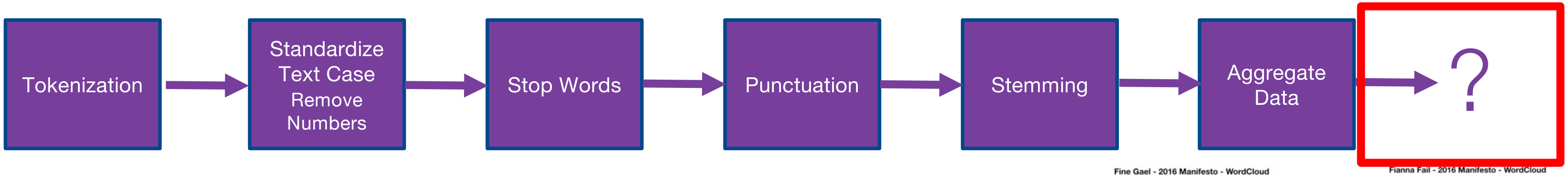
Aggregate Data

?

- Aggregate the Data – Simply counting

```
#get the frequency of each word from collections
from collections import Counter

# count frequencies
cnt = Counter() for word in filtered_words:
   cnt[word] += 1

print(cnt)
```

```
Counter({'new': 340, 'support': 249, 'work': 190, 'public': 186,
'government': 177, 'ensure': 177, 'plan': 176, 'continue': 168, 'local': 150,
...
```



- What do you want to do with this data?

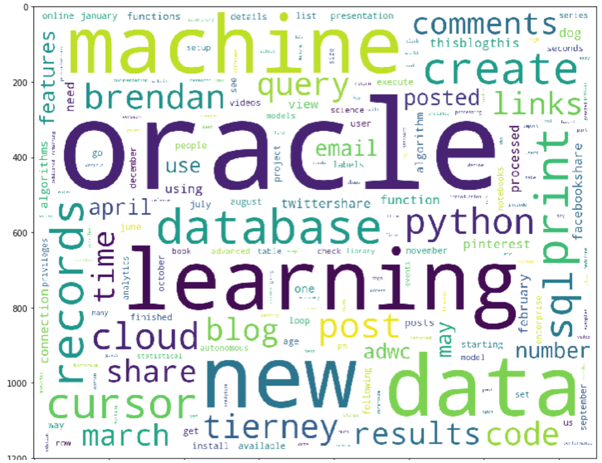
```
#create a word cloud using frequencies for emphasis
from wordcloud import WordCloud
import matplotlib.pyplot as plt

wc = WordCloud(max_words=100, margin=9, background_color='white’,
               scale=3, relative_scaling = 0.5, width=500, height=400,
               random_state=1).generate_from_frequencies(cnt)

plt.figure(figsize=(20,10))
plt.imshow(wc)
plt.show() #Save the image in the img folder:
wc.to_file(wkDir+party+"_2016.png")
```

Fine Gael - 2016 Manifesto - WordCloud

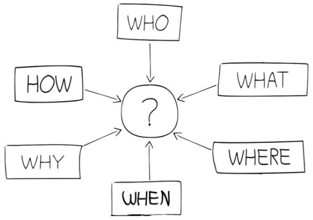
Fianna Fail - 2016 Manifesto - WordCloud

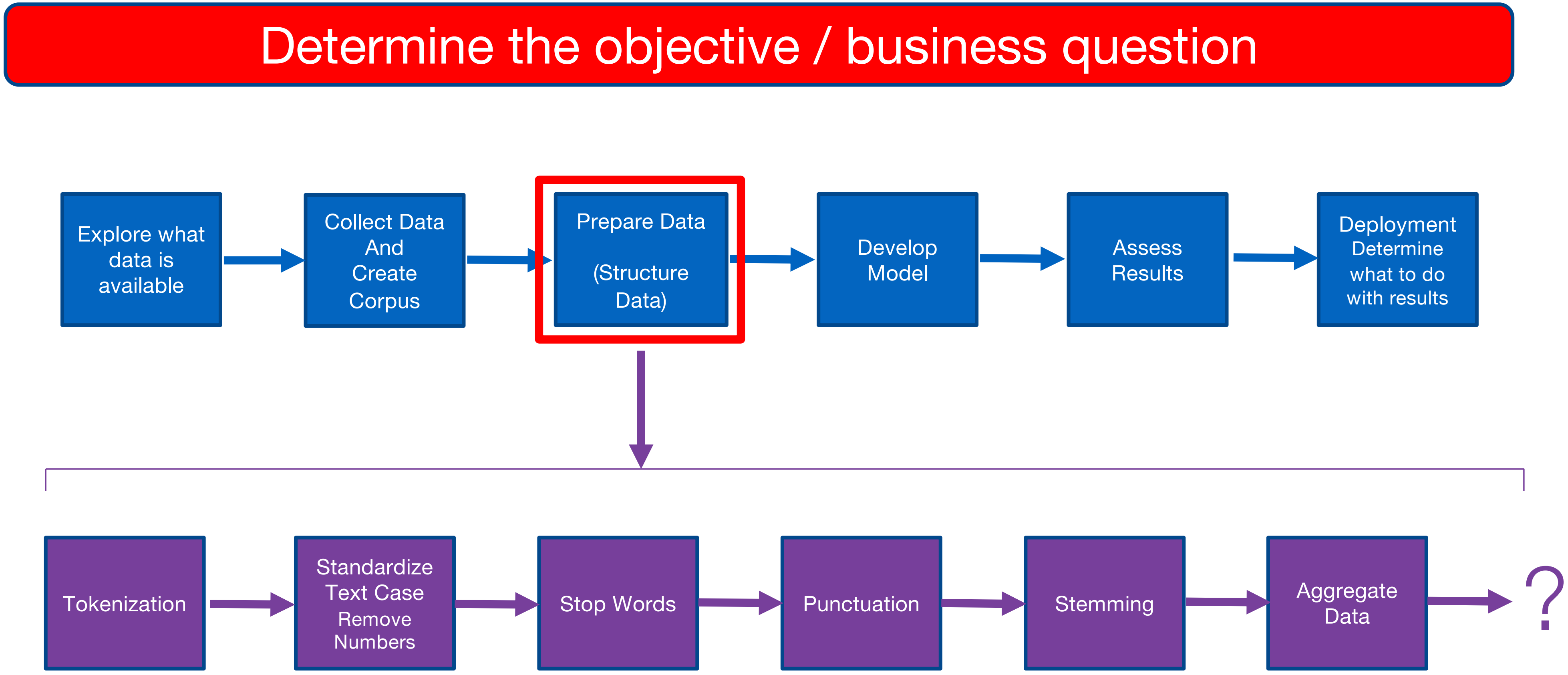
Labour - 2016 Manifesto - WordCloud

Sinn Fein - 2016 Manifesto - WordCloud



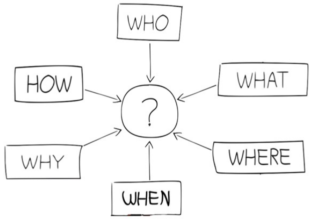
Processing Text – Core Steps



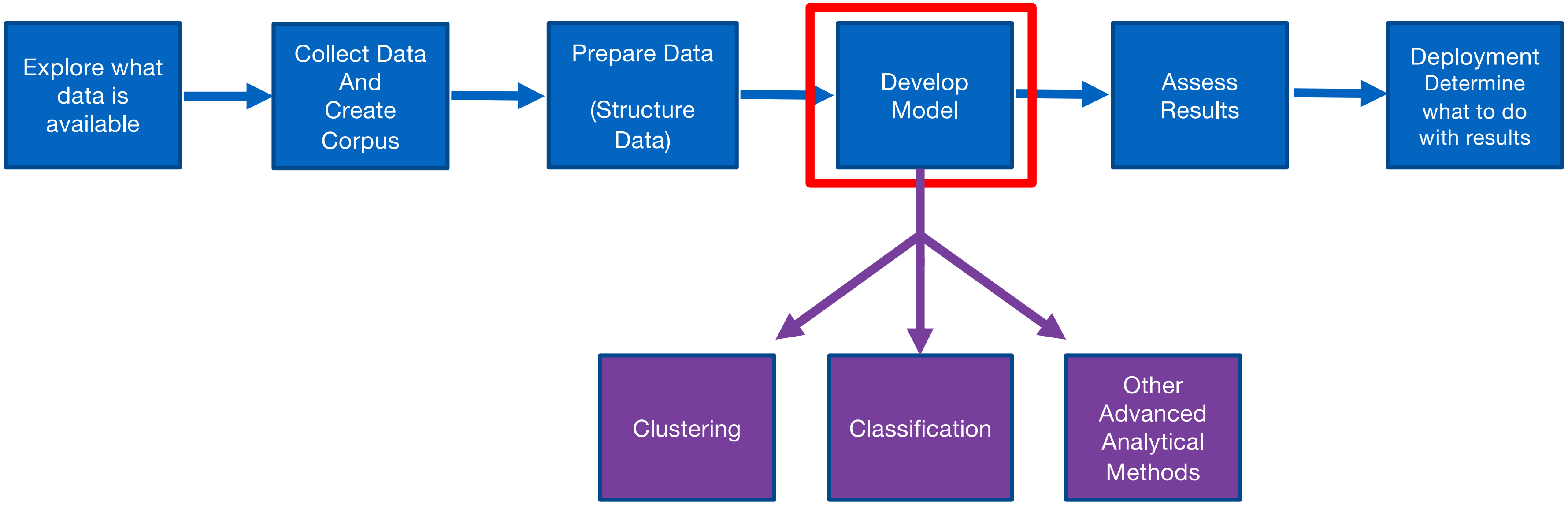

Determine the objective / business question
Explore what data is available
Collect Data And Create Corpus
Prepare Data (Structure Data)
Develop Model
Assess Results
Deployment Determine what to do with results
Tokenization
Standardize Text Case Remove Numbers
Stop Words
Punctuation
Stemming
Aggregate Data
?

Very similar to CRISP-DM process

Processing Text – Core Steps



Determine the objective / business question


Explore what data is available
Collect Data And Create Corpus
Prepare Data (Structure Data)
Develop Model
Assess Results
Deployment Determine what to do with results
Clustering
Classification
Other Advanced Analytical Methods

Very similar to CRISP-DM process

Demo

x2

Term Frequency – Inverse Document Frequency

- Commonly known as TF-IDF is a weighting factor used in applications such as information retrieval and text mining. TF-IDF uses statistics to measure how important a word is to a particular document.

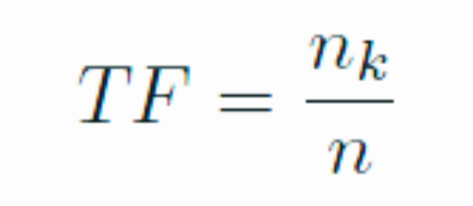
  - **TF — Term Frequency:** measures how frequently a string occurs in a document. Calculated as the total number of occurrences in the document divided by the total length of the document (for normalization).
  - **IDF — Inverse Document Frequency:** measures the importance of a string within a document. For example, certain strings such as “is”, “of”, and “a”, will appear a lot of times in many documents but don’t really hold much meaning—they’re not adjectives or verbs. IDF, therefore, weights each string according to its importance, calculated as the *log()* of the total number of documents in the dataset divided by the number of documents that the string occurs in (+1 in the denominator to avoid a division by zero).
  - **TF-IDF:** The final calculation of the TF-IDF is simply the multiplication of the TF and IDF terms: *TF * IDF*.

- Words that occur more frequently in a document are weighted higher, but only if they’re more rare within the whole document.

- Often used by search engines in scoring and ranking a document’s relevance given a keyword input.

Term Frequency – Inverse Document Frequency

- Calculating TF part is straightforward:



$$TF = \frac{n_k}{n}$$

- Simply ratio of number of times keyword, *k*, appears in given document, n_k, to total number of terms in document, n.

e.g. a common English word such as “*that*” will have fairly high TF score and “*EnterpriseMiner*” will have much lower TF score.

Term Frequency – Inverse Document Frequency

- IDF is defined as follows:

$$IDF = \log_2 \frac{N}{N_k}$$

- where N is number of documents (for search engines, N is number of all indexed web pages). N_k number of documents that contain keyword, k.
- Common word e.g. “that” would have ratio $\frac{N}{N_k} \approx 1$, yielding IDF score ≈ 0.
- “EnterpriseMiner” likely appears in relatively fewer documents, so N/N*k* much greater than 1 i.e. IDF is high for less common keyword.

Term Frequency – Inverse Document Frequency

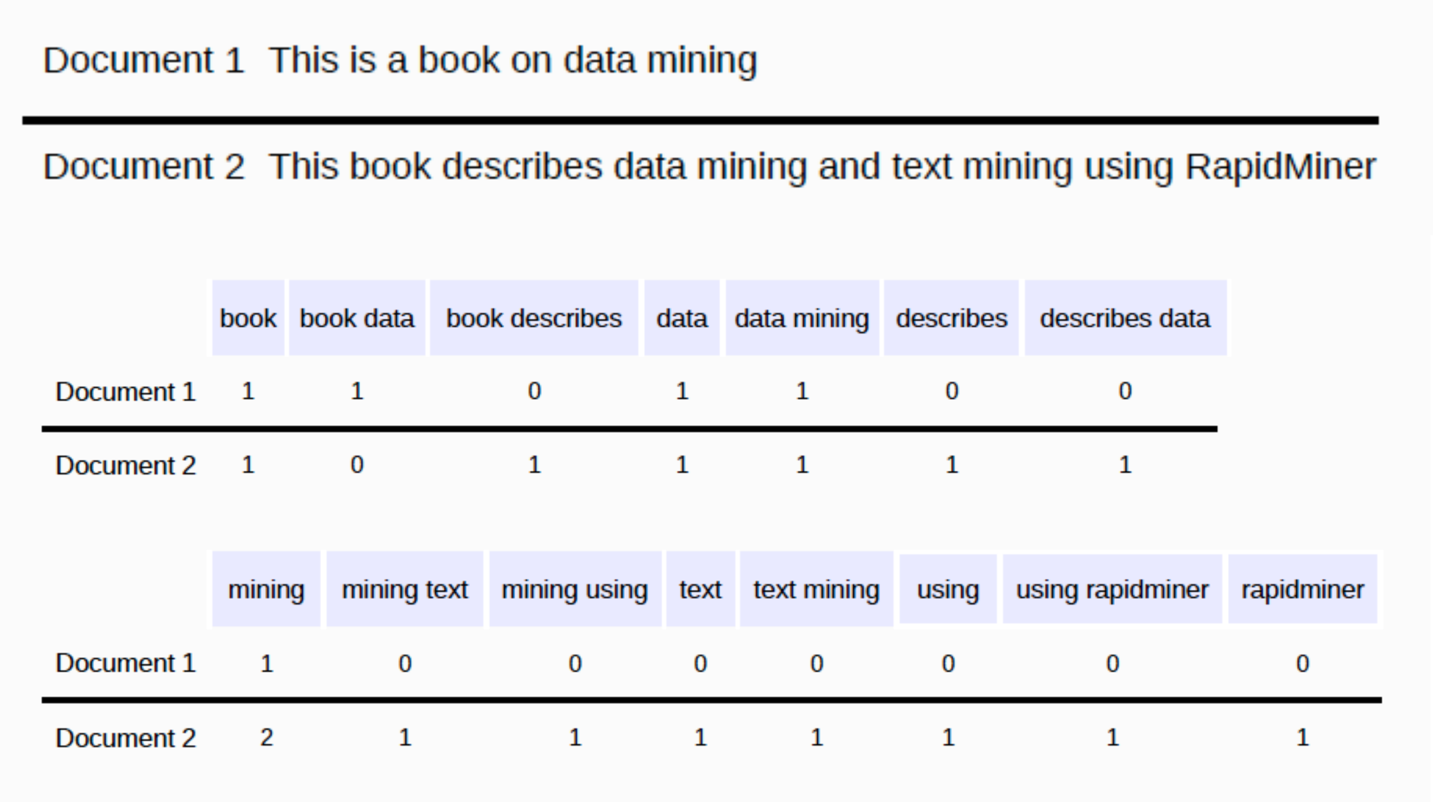
- The final calculation of the TF-IDF is simply the multiplication of the TF and IDF terms: *TF * IDF*.

$$TF - IDF = TF \times IDF$$

N-grams – Multi-Word Features

- In spoken and written language words typically go together
  - e.g. “Good” often followed by either “Morning”, “Afternoon”, “Evening” or “Night”.
- By grouping such terms, called n-grams, and analysing them can provide additional insights - particularly bigrams (2 words) and trigrams (3 words).
- Final pre-processing step typically involves forming these n-grams and storing them in document vector.
- Note: Algorithms providing n-grams are computationally expensive and results can become huge.

N-grams



Document 1	This is a book on data mining
Document 2	This book describes data mining and text mining using RapidMiner

	book	book data	book describes	data	data mining	describes	describes data
Document 1	1	1	0	1	1	0	0
Document 2	1	0	1	1	1	1	1

	mining	mining text	mining using	text	text mining	using	using rapidminer	rapidminer
Document 1	1	0	0	0	0	0	0	0
Document 2	2	1	1	1	1	1	1	1

- TF-based document vector for bigrams

- Note, terms like “data mining” and “text mining” and “using RapidMiner” can be quite meaningful in this context.

These **10 text mining examples** can give you an idea of how this technology is helping organizations today.

1 – Risk management

No matter the industry, Insufficient risk analysis is often a leading cause of failure. This is especially true in the financial industry where adoption of **Risk Management Software** based on text mining technology can **dramatically increase the ability to mitigate risk**, enabling complete management of thousands of sources and petabytes of text documents, and providing the ability to link together information and be able to access the right information at the right time.

2 – Knowledge management

Not being able to find important information quickly is always a challenge when managing large volumes of text documents—just ask anyone in the healthcare industry. Here, **organizations are challenged with a tremendous amount of information**—decades of research in genomics and molecular techniques, for example, as well as volumes of clinical patient data—that could potentially be useful for their largest profit center: new product development. Here, **knowledge management software based on text mining offer a clear and reliable solution** for the “info-glut” problem.

3 – Cybercrime prevention

The anonymous nature of the internet and the many communication features operated through it contribute to the increased risk of internet-based crimes. Today, **text mining intelligence and anti-crime applications are making internet crime prevention easier** for any enterprise and law enforcement or intelligence agencies.

4 – Customer care service

Text mining, as well as natural language processing are frequent applications for customer care. Today, text analytics software is frequently adopted to **improve customer experience** using different sources of valuable information such as surveys, trouble tickets, and customer call notes to improve the quality, effectiveness and speed in resolving problems. **Text analysis is used to provide a rapid, automated response to the customer**, dramatically **reducing their reliance on call center** operators to solve problems.

5 – Fraud detection through claims investigation

Text analytics is a tremendously effective technology in any domain where the majority of information is collected as text. Insurance companies are taking advantage of text mining technologies by **combining the results of text analysis with structured data to prevent frauds** and swiftly process claims.

https://analyticsweek.com/content/10-text-mining-examples/

6 – Contextual Advertising

Digital advertising is a moderately new and growing field of application for text analytics. Compared to the traditional cookie-based approach, contextual advertising provides better accuracy, completely preserves the user's privacy.

7 – Business intelligence

This process is used by large companies to uphold and support decision making. Here, **text mining really makes the difference, enabling the analyst to quickly jump at the answer** even when analyzing petabytes of internal and open source data. Applications such as the Cogito Intelligence Platform are able to monitor thousands of sources and analyze large data volumes to extract from them only the relevant content.

8 – Content enrichment

While it's true that working with text content still requires a bit of human effort, text analytics techniques make a significant difference when it comes to being able to more effectively manage large volumes of information. **Text mining techniques enrich content, providing a scalable layer to tag**, organize and summarize the available content that makes it suitable for a variety of purposes.

9 – Spam filtering

E-mail is an effective, fast and reasonably cheap way to communicate, but it comes with a dark side: spam. Today, **spam is a major issue for internet service providers**, increasing their costs for service management and hardware software updating; for users, spam is an entry point for viruses and impacts productivity. **Text mining techniques can be implemented to improve the effectiveness of statistical-based filtering methods**.

10 – Social media data analysis

Today, **social media is one of the most prolific sources of unstructured data**; organizations have taken notice. Social media is increasingly being recognized as a valuable source of market and customer intelligence, and companies are using it to analyze or predict customer needs and understand the perception of their brand. In both needs **text analytics can address both by analyzing large volumes of unstructured data, extracting opinions, emotions and sentiment and their relations with brands and products**.

https://analyticsweek.com/content/10-text-mining-examples/

Advanced Topics

- Machine Translation
- Language Translation
- Language structure
- Q&A Applications
- Resolving Authorship of text, music, etc
- Fake News
- Automatic Speech Writing
- Automatic Generated Content
- Spam Filtering
- Ambiguity Detection and Correction

Any Questions ?

What Now/Next ?