COMP41680

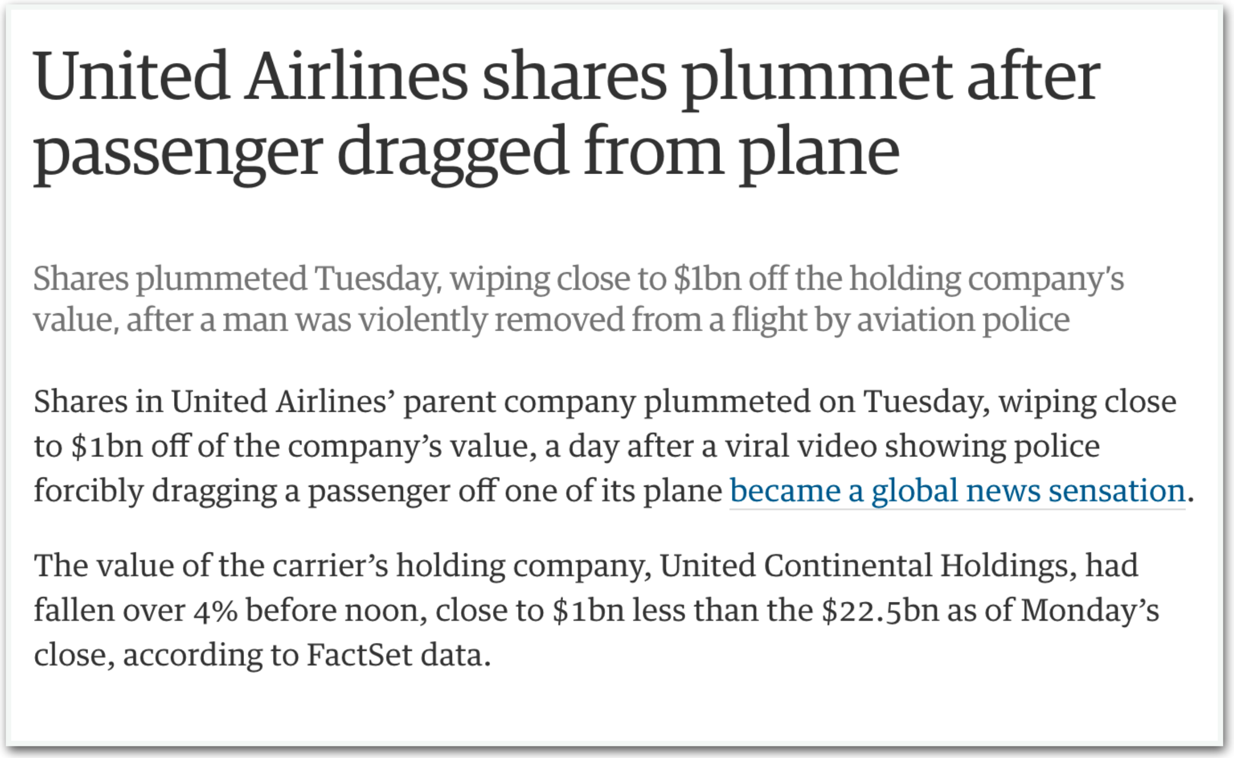
Text Mining

Derek Greene

UCD School of Computer Science
Spring 2023

Unstructured Text

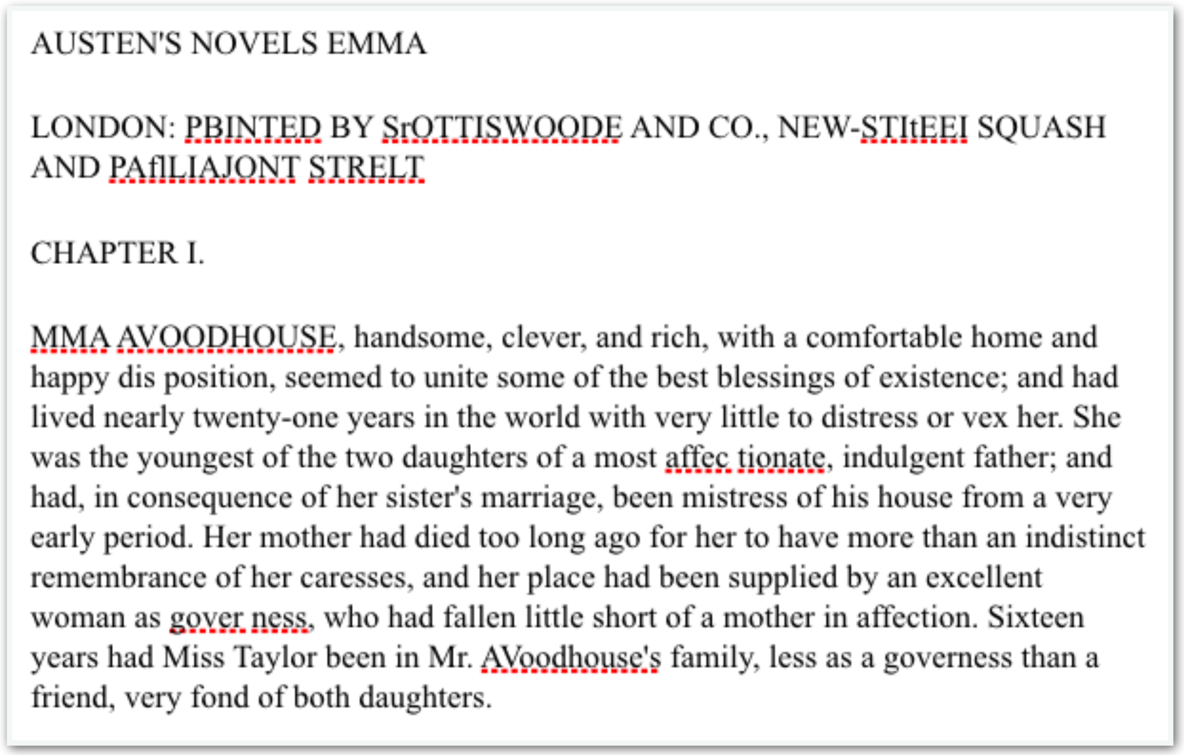
Most textual data arrives in an unstructured form without any pre-defined organisation or format, beyond natural language. The vocabulary, formatting, and quality of the text can vary significantly.


United Airlines shares plummet after passenger dragged from plane

Shares plummeted Tuesday, wiping close to $1bn off the holding company's value, after a man was violently removed from a flight by aviation police

Shares in United Airlines' parent company plummeted on Tuesday, wiping close to $1bn off of the company's value, a day after a viral video showing police forcibly dragging a passenger off one of its plane became a global news sensation.

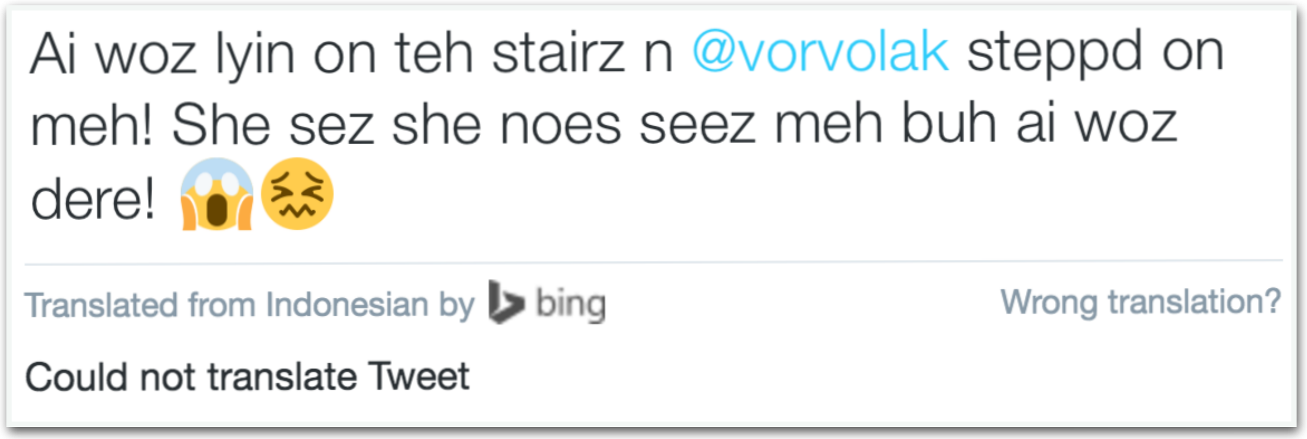
The value of the carrier's holding company, United Continental Holdings, had fallen over 4% before noon, close to $1bn less than the $22.5bn as of Monday's close, according to FactSet data.


AUSTEN'S NOVELS EMMA

LONDON: PBINTED BY SrOTTISWOODE AND CO., NEW-STItEEI SQUASH AND PAfILIAJONT STRELT

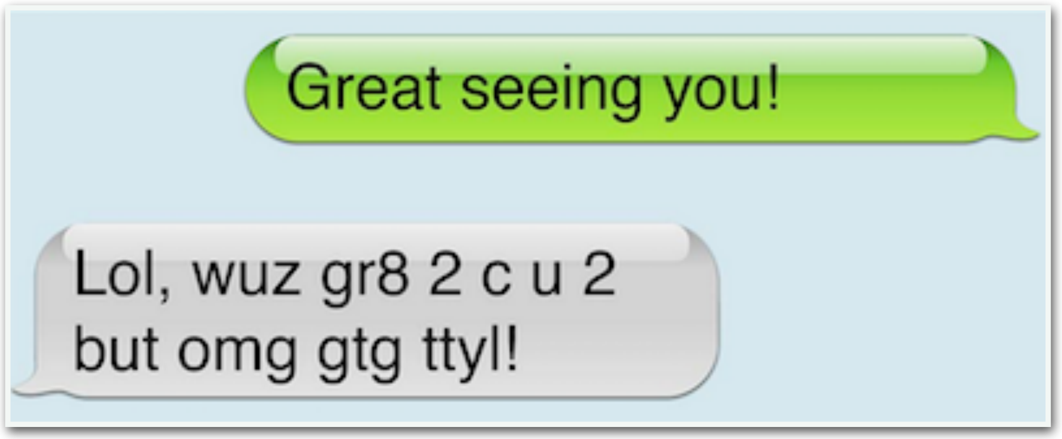
CHAPTER I.

MMA AVOODHOUSE, handsome, clever, and rich, with a comfortable home and happy dis position, seemed to unite some of the best blessings of existence; and had lived nearly twenty-one years in the world with very little to distress or vex her. She was the youngest of the two daughters of a most affec tionate, indulgent father; and had, in consequence of her sister's marriage, been mistress of his house from a very early period. Her mother had died too long ago for her to have more than an indistinct remembrance of her caresses, and her place had been supplied by an excellent woman as gover ness, who had fallen little short of a mother in affection. Sixteen years had Miss Taylor been in Mr. AVoodhouse's family, less as a governess than a friend, very fond of both daughters.


Ai woz lyin on teh stairz n @vorvolak steppd on meh! She sez she noes seez meh buh ai woz dere! 😱😣

Translated from Indonesian by bing — Wrong translation?

Could not translate Tweet


Great seeing you!

Lol, wuz gr8 2 c u 2 but omg gtg ttyl!

Common Tasks in Text Mining

- *Text classification*: e.g. Is a new email spam or non-spam?
- *Topic modelling*: e.g. What are the main subjects being discussed around EU Brexit negotiations?
- *Sentiment analysis*: e.g. Are Twitter users talking positively or negatively about the new iPhone?
- *Review mining*: e.g. Can we extract the most positive and negative features in reviews for a given hotel on TripAdvisor?
- *Authorship attribution*: e.g. Did Shakespeare write all his own dramas?
- *Genre classification*: e.g. Can we automatically assign a new novel to an Amazon genre category?
- *Moderation*: e.g. Can we identify potentially abusive or offensive posts on forums or on social media platforms?
- *Plagiarism detection*: e.g. Are two submitted reports unusually similar?

Tokenising Text

- Raw text documents are textual, not numeric. The first step in analysing unstructured documents is to split the raw text into individual tokens, each corresponding to a single term (word).
- In the simplest case we might split a text string by whitespace:

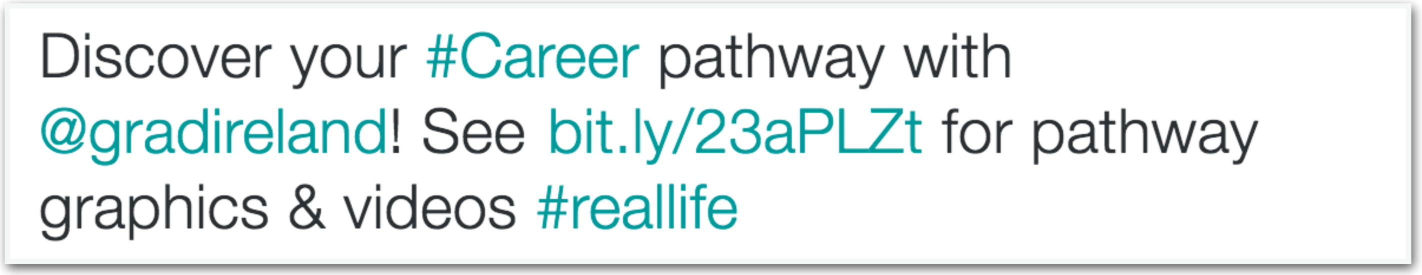
```
text = "IMF cuts global growth outlook"
text.split(" ")
```

```
['IMF', 'cuts', 'global', 'growth', 'outlook']
```

- In most cases we need to deal with other types of punctuation:

```
Bad-news ahead? The IMF cuts outlook, says:'growth in doubt…'
```

- For some types of text, certain characters can have significance:



Discover your #Career pathway with @gradireland! See bit.ly/23aPLZt for pathway graphics & videos #reallife

Tokenising Text

- Scikit-learn provides intelligent tokenisation functions for dealing with English text - i.e. converting a string to a list of tokens.

```
from sklearn.feature_extraction.text import CountVectorizer
tokenize = CountVectorizer().build_tokenizer()
```

```
text = "IMF cuts global growth outlook"
tokenize(text)
```

```
['IMF', 'cuts', 'global', 'growth', 'outlook']
```

```
text = "Bad-news ahead? The IMF cuts outlook, says:Growth in doubt"
tokenize(text)
```

```
['Bad', 'news', 'ahead', 'The', 'IMF', 'cuts', 'outlook', 'says', 'Growth', 'in', 'doubt']
```

- Punctuation and single letter tokens are removed. A standard tokeniser is not always suitable…

```
tweet = "Discover your #Career pathway with @gradireland! See http://bit.ly/23aPLZt"
tokenize(tweet)
```

```
['Discover', 'your', 'Career', 'pathway', 'with', 'gradireland', 'See', 'http', 'bit',
'ly', '23aPLZt']
```

Counting Term Frequencies

- Once we have performed tokenisation, we might then count the frequency of occurrence of terms (tokens) in each document.

Growth forecasts cut as IMF warns of risk. The IMF cut its global growth forecast for 2022. It warned of widespread global stagnation risk and stated the global economy could be vulnerable in 2022.

Term	Frequency
global	3
risk	2
cut	2
IMF	2
2022	2
...	...

- We can repeat this process to compute frequencies across an entire corpus and the sum the frequencies.

Growth forecasts cut as IMF warns of risk. The IMF
c
v
g

IMF chief economist Maurice Obstfeld said in a
sta
eff
the

The Fund called on global policymakers attending the IMF and World Bank meetings, being held in Washington, to take coordinated actions to boost demand with structural economic reforms in 2022.

Term	Frequency
global	5
imf	4
growth	3
2022	3
economy	3
...	...

Bag-of-Words Model for Documents

- How can we go from tokens to numeric features?
- Bag-of-words model: Each document is represented by a vector in a *m*-dimensional coordinate space, where *m* is total number of unique terms across all documents (the corpus vocabulary).

Document 1:
Forecasts cut as IMF issues warning

Document 2:
IMF and WBG meet to discuss economy

Document 3:
WBG issues 2022 growth warning

Example:

When we tokenise our corpus of 3 documents, we have a vocabulary of 14 distinct terms (when no filtering is applied).

2022
and
as
cut
discuss
economy
Forecasts
growth
IMF
issues
meet
to
warning
WBG

Bag-of-Words Representation

- Each document can be represented as a term vector, with an entry indicating the number of times a term appears in the document:

Document 1:
Forecasts cut as IMF issues warning

2022	Forecasts	IMF	WBG	and	as	cut	discuss	economy	growth	issues	meet	to	warning
0	1	1	0	0	1	1	0	0	0	1	0	0	1

- By transforming all documents in this way, and stacking them in rows, we create a full document-term matrix:

Document 2:
IMF and WBG meet to discuss economy

Document 3:
2022: WBG issues 2022 growth warning

2022	Forecasts	IMF	WBG	and	as	cut	discuss	economy	growth	issues	meet	to	warning
0	1	1	0	0	1	1	0	0	0	1	0	0	1
0	0	1	1	1	0	0	1	1	0	0	1	1	0
2	0	0	1	0	0	0	0	0	1	1	0	0	1

3 Documents x 14 Terms

Bag-of-Words Representation

- The position and context of terms within the original document text is lost when using this model.

Document 1:
Forecasts cut as IMF issues warning

2022	Forecasts	IMF	WBG	and	as	cut	discuss	economy	growth	issues	meet	to	warning
0	1	1	0	0	1	1	0	0	0	1	0	0	1

- The size of the vocabulary is often large, meaning that the resulting vectors can be very high dimensional.
- Fortunately, most values in the document-term matrix will be zeros, since for a given document less than a couple thousands of distinct terms will be used.
- Bag-of-word models are typically high-dimensional sparse datasets. Document-term matrix often has > 98% zero values.

Bag-of-Words in Python

- Scikit-learn includes functionality to easily transform a collection of strings containing documents into a document-term matrix.

The input is a list of strings. Each string is a separate document.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)
```

The output *X* is a sparse NumPy 2D array, with rows corresponding to documents and columns corresponding to terms.

- Once the matrix has been created, we can access the list of all terms and an associated dictionary (`vocabulary_`) which maps each unique term to a corresponding column in the matrix.

```
terms = vectorizer.get_feature_names()
len(terms)
```

```
3288
```

How many terms in the vocabulary?

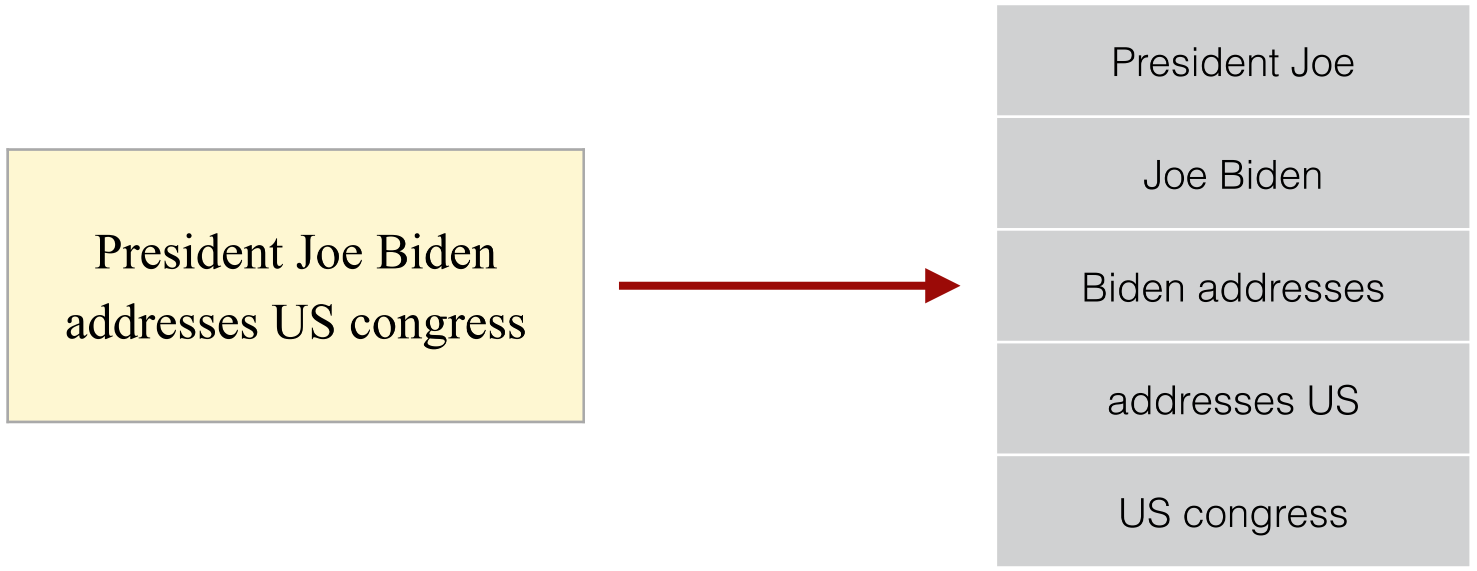
```
vocab = vectorizer.vocabulary_
vocab["world"]
```

```
3246
```

Which column corresponds to term?

N-Grams

- A bag-of-words model does not preserve sequence information, so the order of words in a document is lost.
- **Solution:** Build terms using sequences of adjacent tokens. Then construct a document-term matrix in the same way.
- Term bigrams: Build terms from every pair of adjacent tokens.



Document can be represented as bag-of-words model with 5 term bigrams

- Term *n*-grams: Build terms from *n* adjacent tokens.
- Disadvantage: This approach significantly increases the size of the vocabulary for a given corpus.

N-Grams in Python

- We can use Scikit-learn's CountVectorizer to create a document-term matrix with term *n*-grams representing each document.
- We specify an extra argument `ngram_range` which specifies the shortest and longest token sequences to include. Length 1 is just a single token.

Transform our input documents, extracting single tokens and bigrams (*n=1* and *n=2*):

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(ngram_range = (1,2))
X = vectorizer.fit_transform(documents)
```

Display some sample terms - we see single tokens and bigrams (phrases of length 2):

```
terms = vectorizer.get_feature_names()
print(terms[510:520])
```

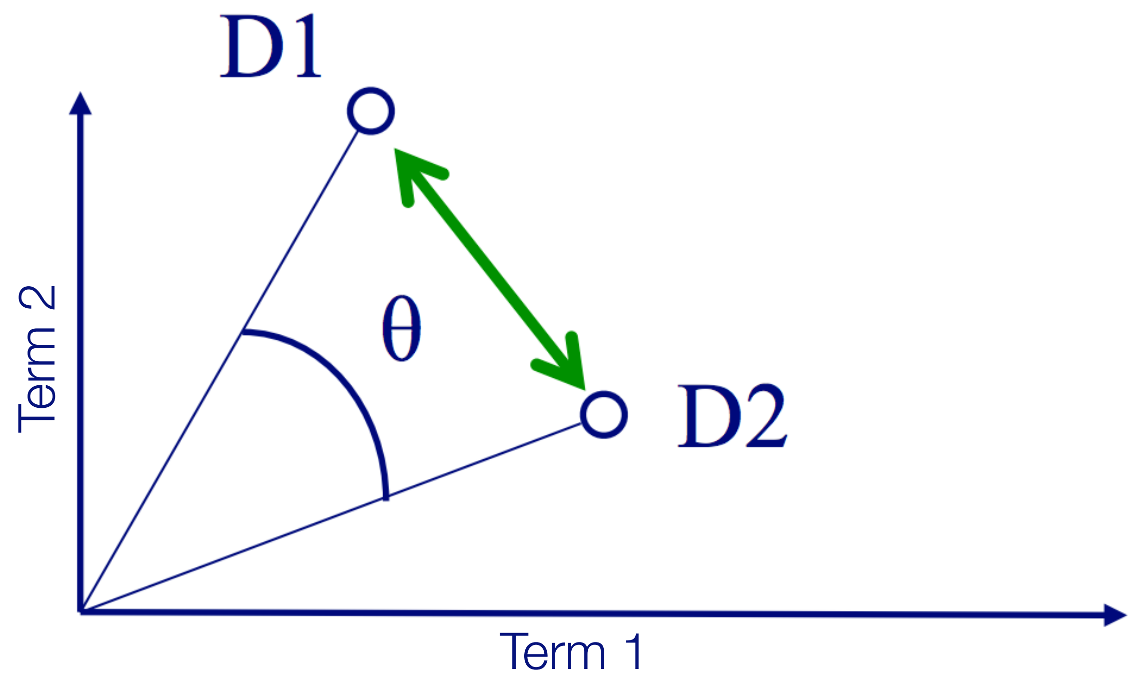
```
['admitted victory', 'adriatic', 'adriatic coast', 'advanced',
'advanced position', 'advantage', 'advantage and', 'advantage
of', 'adverts', 'adverts for']
```

Measuring Similarity

- Cosine similarity: Most common approach for measuring similarity between two documents in a bag-of-words representation is to look at the cosine of the angle between their term vectors.
- Motivation: vectors for documents containing similar terms will point in the same direction in the *m*-dimensional vector space.

$$cos(D1, D2) = \frac{D1 \cdot D2}{||D1||\ ||D2||}$$

$$\text{where} \quad ||D|| = \sqrt{\sum_{i=1}^{m} d_i^2}$$



- Cosine similarity score is 1 if two documents are identical, 0 if two documents share no terms in common.

Measuring Similarity

Document 1:
Forecasts cut as IMF issues warning

Document 2:
IMF and WBG meet to discuss economy

Document 3:
IMF and WBG meet to discuss growth

and	as	cut	discuss	economy	Forecasts	growth	IMF	issues	meet	to	warning	WBG
0	1	1	0	0	1	0	1	1	0	0	1	0
1	0	0	1	1	0	0	1	0	1	1	0	1
1	0	0	1	0	0	1	1	0	1	1	0	1

$$cos(D1, D2) = 0.15 \quad cos(D1, D3) = 0.15$$

$$cos(D2, D3) = 0.86$$

We can perform the same calculations in Python...

```
from sklearn.metrics.pairwise import cosine_similarity
print( "cos(D1,D2) = %.2f" % cosine_similarity( X[0], X[1] ) )
print( "cos(D1,D3) = %.2f" % cosine_similarity( X[0], X[2] ) )
print( "cos(D2,D3) = %.2f" % cosine_similarity( X[1], X[2] ) )
```

```
cos(D1,D2) = 0.15
cos(D1,D3) = 0.15
cos(D2,D3) = 0.86
```

Measure similarity between rows of the document-term matrix X

Challenges in Text Mining

- The Sparsity Problem: Natural language use often involves huge vocabularies, so most documents will share very few words.
- The use of *n*-grams also increases sparsity, as even fewer documents will share the same phrases.
- This problem is exacerbated due to…
  - Synonymy: languages often use many different words to refer to identical or closely related concepts (e.g. ‘residence’,‘abode’).
  - Homonymy: a single word can have multiple **unrelated** meanings (e.g. ‘jaguar’ - car or animal?).
  - Polysemy: a single word can have multiple **related** meanings (e.g. ‘bank’ - a financial institution or the building housing it?).
- This applies when analysing a monolingual corpus. Text analysis becomes far more difficult when working with a multilingual corpus.

Challenges in Text Mining

- The Sparsity Problem has several practical consequences...
  - Additional memory and processing requirements due to more dimensions (terms).
  - Analytical problems related to sparsity - e.g. when using a bag-of-words representation, we can fail to identify documents which are related to the same concepts.

Document 1:
jaguars are expensive cars

Document 2:
a jaguar is a costly vehicle

Document 3:
the jaguar, a feline animal

a	animal	are	cars	costly	expensive	feline	is	jaguar	jaguars	the	vehicle
0	0	1	1	0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	1	0	0	0	0	1	0	1	0	1	0

$cos(D1, D2) = 0.0$ $cos(D2, D3) = 0.4$

Text Preprocessing

- The number of terms used to represent documents (and hence the sparsity) is often reduced by applying a number of simple preprocessing techniques before building a document-term matrix:
    - Case conversion: Converting all terms to lowercase.
    - Minimum term length: Exclude terms of length < 2
    - Stemming: Process by which endings are removed from terms in order to remove things like tense or plurals: e.g. `compute, computing, computer = comput`
    - Lemmatisation: Process to reduce a term to its canonical form. This is a more advanced form of stemming.
    - Stop-word filtering: Remove terms that appear on a pre-defined filter list of terms that are highly frequent and do not convey useful information (e.g. `and, the, while`)
    - Low frequency term filtering: Remove terms that appear in very few documents in a corpus.

Text Preprocessing in Python

- By default Scikit-learn converts tokens to lowercase and removes tokens of length 1 (i.e. single letters).
- Scikit-learn allows us to perform other simple preprocessing steps by passing input arguments to `CountVectorizer`.

```
vectorizer = CountVectorizer(stop_words="english")
X = vectorizer.fit_transform(documents)
```

Use built-in stop-words by specifying the language

```
mywords = [ "and", "the", "am", "pm" ]
vectorizer = CountVectorizer(stop_words=mywords)
X = vectorizer.fit_transform(documents)
```

Use a custom list of stopwords

Filter terms appearing in less than 5 documents

```
vectorizer = CountVectorizer(min_df = 5)
X = vectorizer.fit_transform(documents)
```

Specify multiple preprocessing steps

```
vectorizer = CountVectorizer(stop_words="english", min_df = 5)
X = vectorizer.fit_transform(documents)
```

Text Preprocessing in Python

- To stem or lemmatise tokens, we need to use functionality from another text analysis library: NLTK (http://www.nltk.org)

Install via command line

```
pip install nltk
```

```
conda install nltk
```

- Apply stemming to individual words:

```
from nltk.stem.porter import PorterStemmer
words = ['cycles', 'insists', 'computer', 'flying', 'cars']
stemmer = PorterStemmer()
for w in words:
    print( stemmer.stem(w) )
```

```
cycl insist comput fli car
```

- Apply lemmatisation to individual words:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
for w in words:
    print( lemmatizer.lemmatize(w) )
```

```
cycle insists computer flying car
```

Term Weighting

- As well as including or excluding terms, we can also modify or weight the frequency values themselves.
- We can improve the usefulness of the document-term matrix by giving higher weights to more "important" terms.
- TF-IDF: Common approach for weighting the score for a term in a document. Several different formulations, but always consist of:
  - Term Frequency (TF): Number of times a given term appears in a single document.
  - Inverse Document Frequency (IDF): Function of total number of distinct documents containing a term. Effect is to penalise common terms that appear in almost every document.

Common version is log-based TF-IDF

$$w(t, D) = \underbrace{tf(t, d)}_{\text{TF}} \times \underbrace{(log(\frac{n}{df(t)}) + 1)}_{\text{IDF}}$$

n = total number of documents

Term Weighting

- **Example:** In a corpus of *n*=1000 documents, for a document *D*...

  Term 'cat' appears in the document *D* 3 times, and appears in 50 documents in total.

  Term 'dog' appears in the document *D* 4 times, and appears in 250 documents in total.

$$w(\text{cat}, D) = 3 \times (log(\frac{1000}{50}) + 1) = 11.987$$

$$w(\text{dog}, D) = 4 \times (log(\frac{1000}{250}) + 1) = 9.545$$

Term 'cat' is more "important"

- In Scikit-learn, we can generate a TF-IDF weighted document-term matrix by using `TfidfVectorizer` in place of `CountVectorizer`:

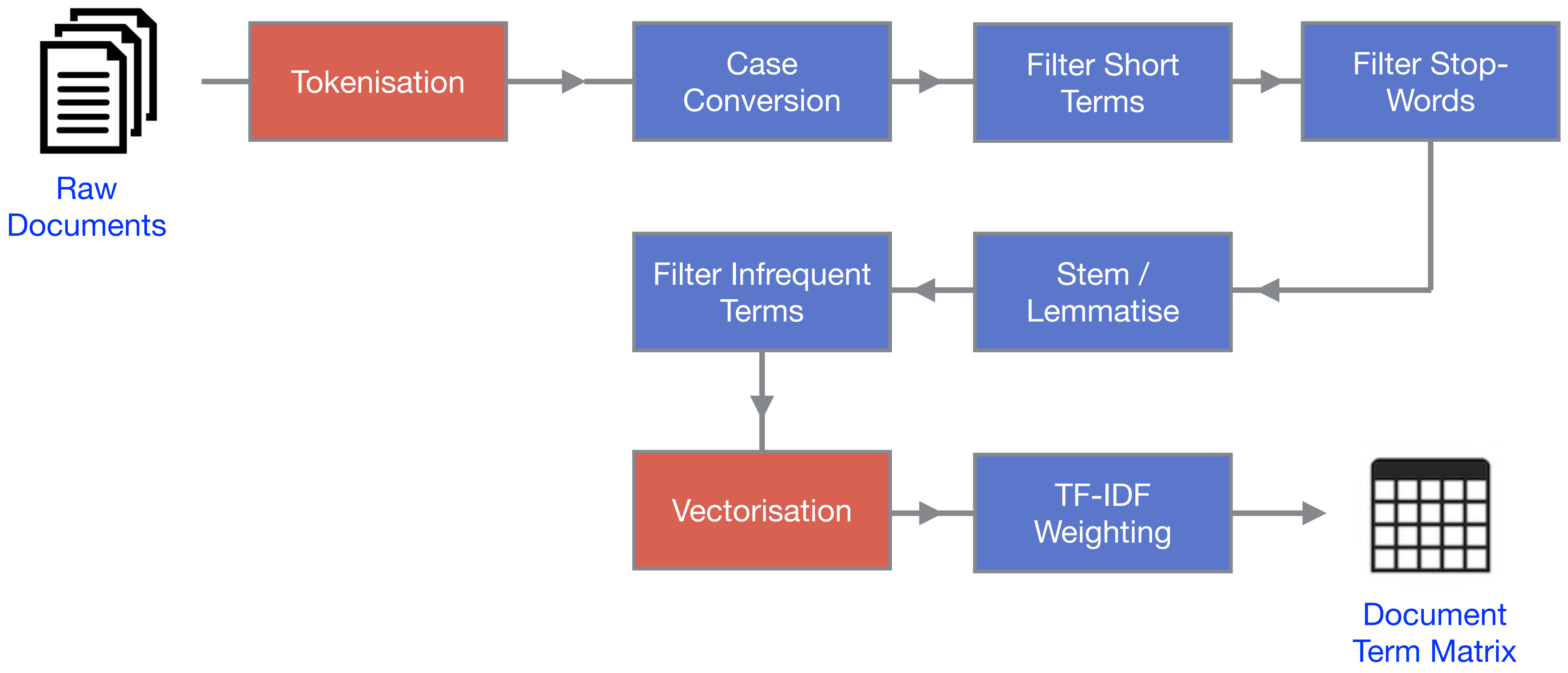
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(documents)
```

```
vectorizer = TfidfVectorizer(stop_words="english", min_df = 5)
X = vectorizer.fit_transform(documents)
```

Can include preprocessing steps as before

Text Processing Pipeline Overview

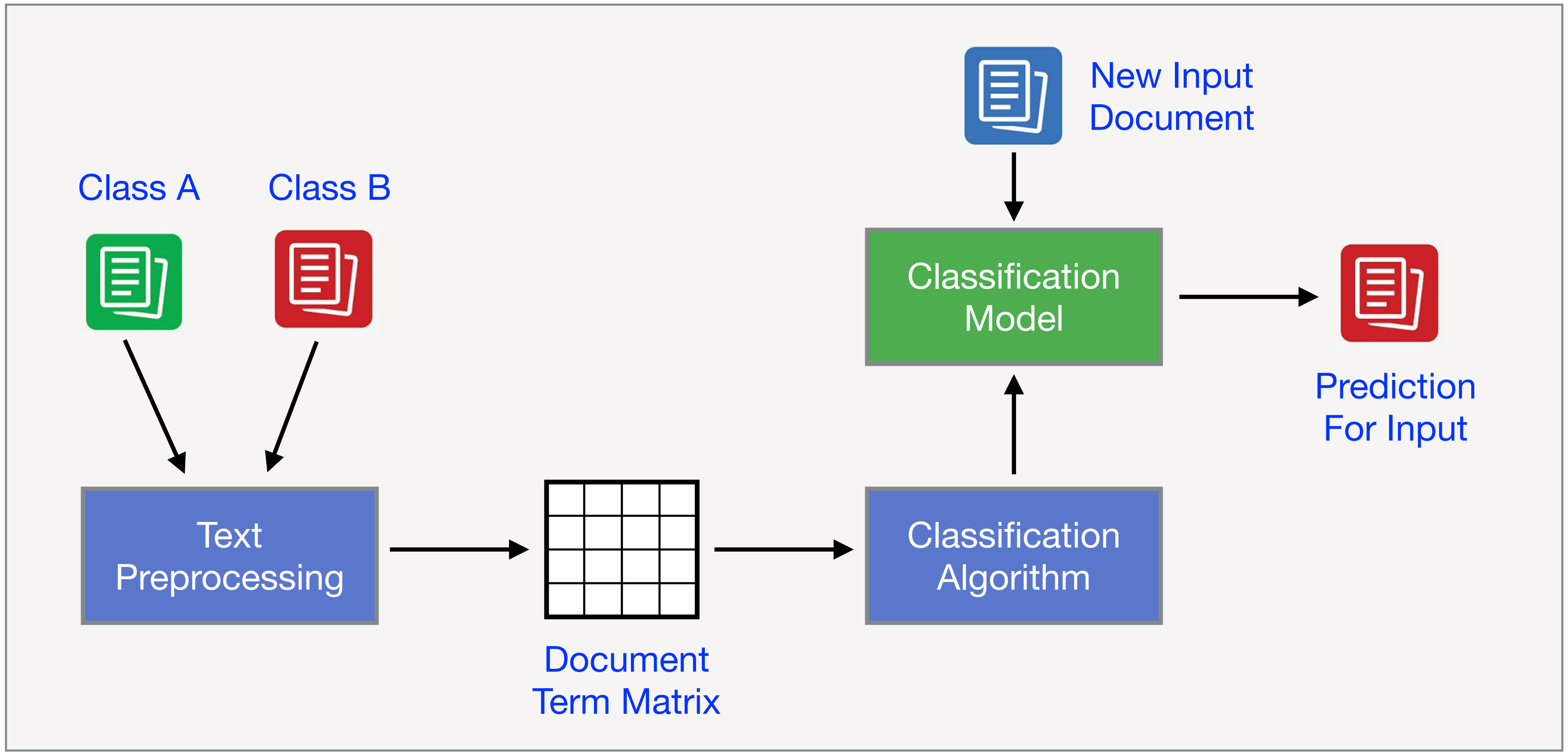
Typical text preprocessing steps for processing a corpus...



Subsequent analysis methods can then be applied to the resulting document-term matrix - e.g. document classification.

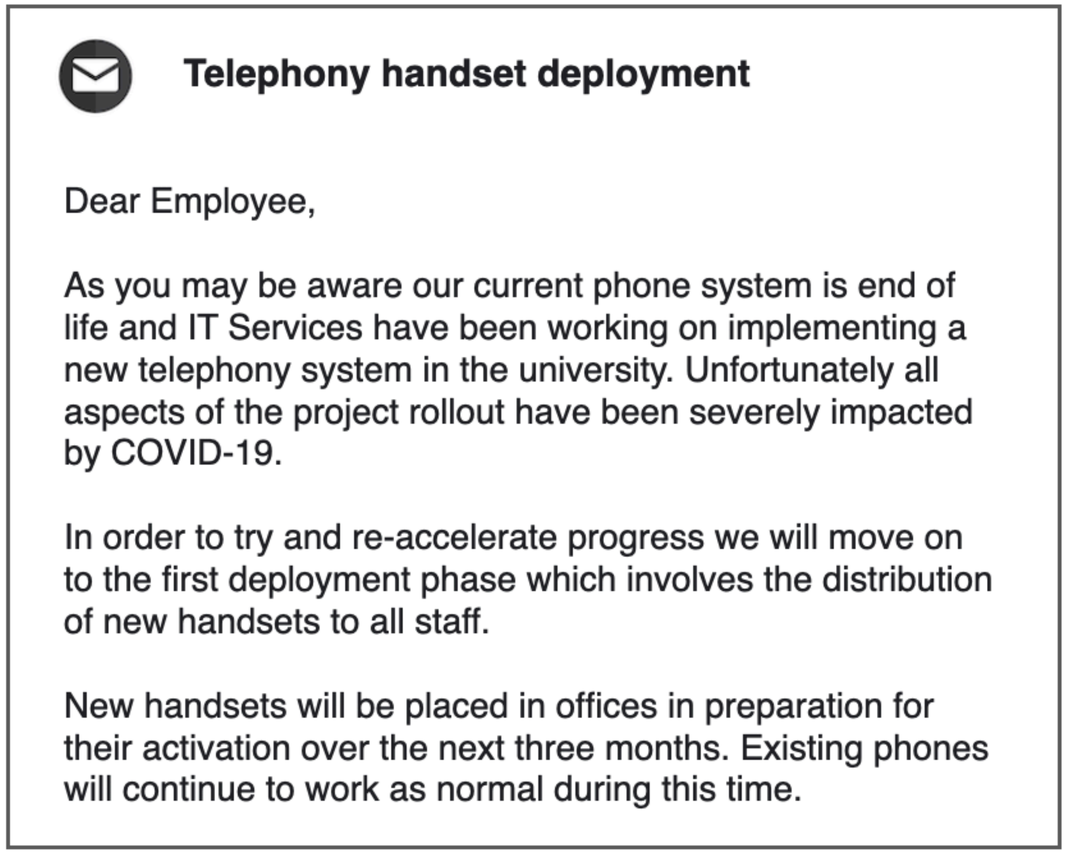
Text Classification

- **Goal:** Learn a model from a training set of documents so we can accurately predict classes for new unlabeled documents.
- **Input:** Training set of labelled text documents, annotated with two or more target class labels.



Text Classification

- **Example: Spam email classification** tries to distinguish between spam and non-spam emails based on their textual content.
  1. Build a binary classification model based on labelled training examples - i.e. previous *spam* or *non-spam* emails.
  2. Apply the model to classify new emails as *spam* or *non-spam*.



Telephony handset deployment

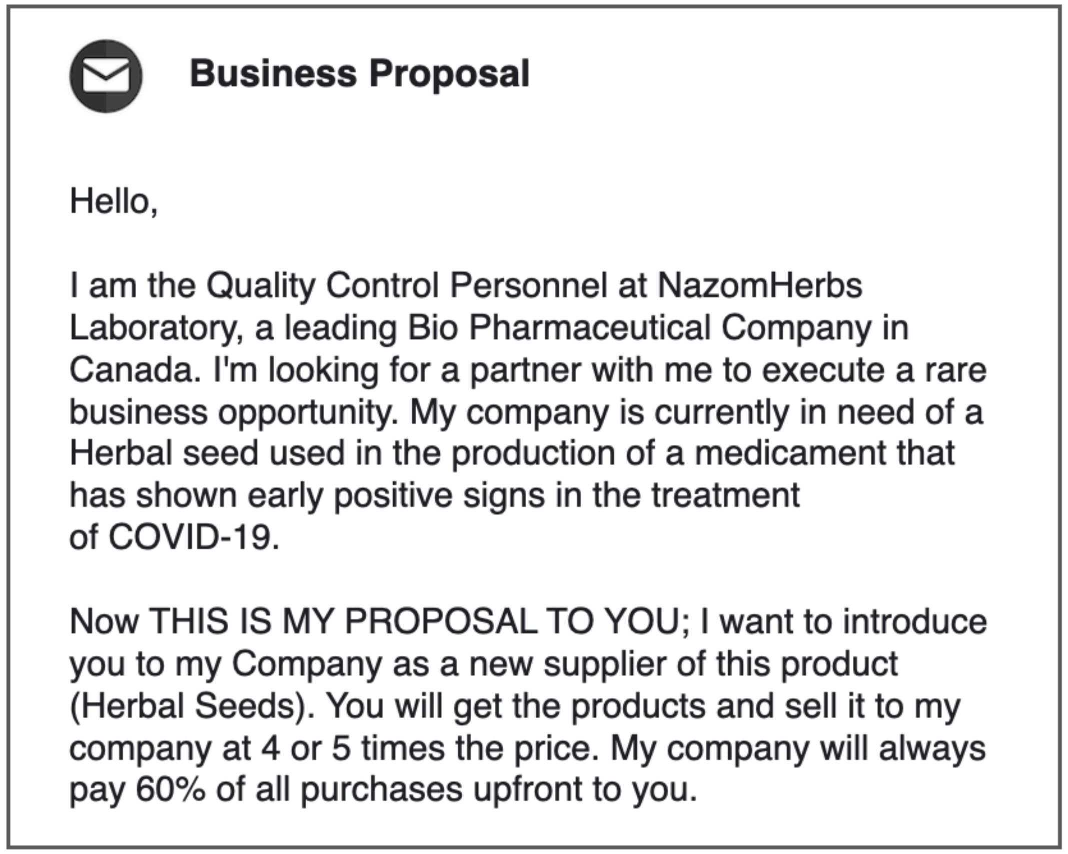
Dear Employee,

As you may be aware our current phone system is end of life and IT Services have been working on implementing a new telephony system in the university. Unfortunately all aspects of the project rollout have been severely impacted by COVID-19.

In order to try and re-accelerate progress we will move on to the first deployment phase which involves the distribution of new handsets to all staff.

New handsets will be placed in offices in preparation for their activation over the next three months. Existing phones will continue to work as normal during this time.

Training example: *Non-spam* class



Business Proposal

Hello,

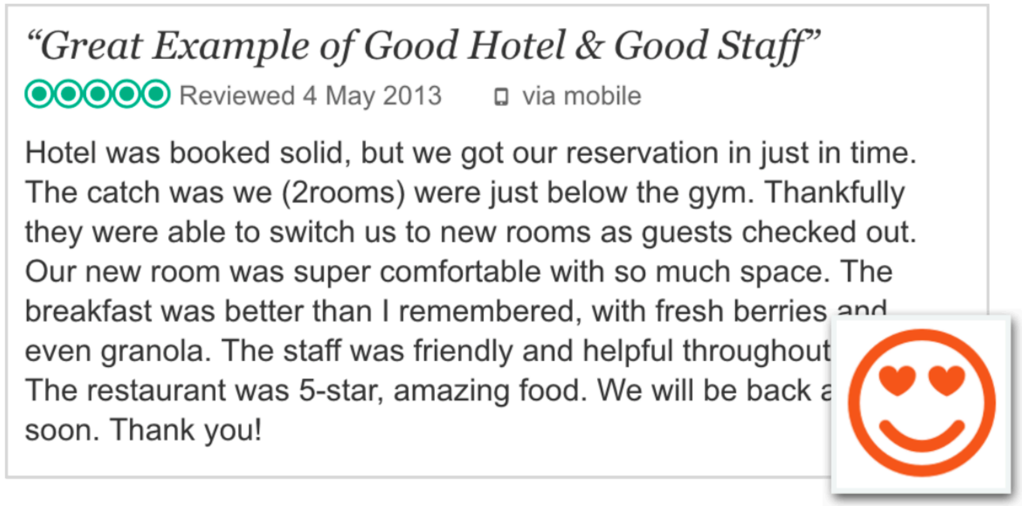
I am the Quality Control Personnel at NazomHerbs Laboratory, a leading Bio Pharmaceutical Company in Canada. I'm looking for a partner with me to execute a rare business opportunity. My company is currently in need of a Herbal seed used in the production of a medicament that has shown early positive signs in the treatment of COVID-19.

Now THIS IS MY PROPOSAL TO YOU; I want to introduce you to my Company as a new supplier of this product (Herbal Seeds). You will get the products and sell it to my company at 4 or 5 times the price. My company will always pay 60% of all purchases upfront to you.

Training example: *Spam* class

Text Classification

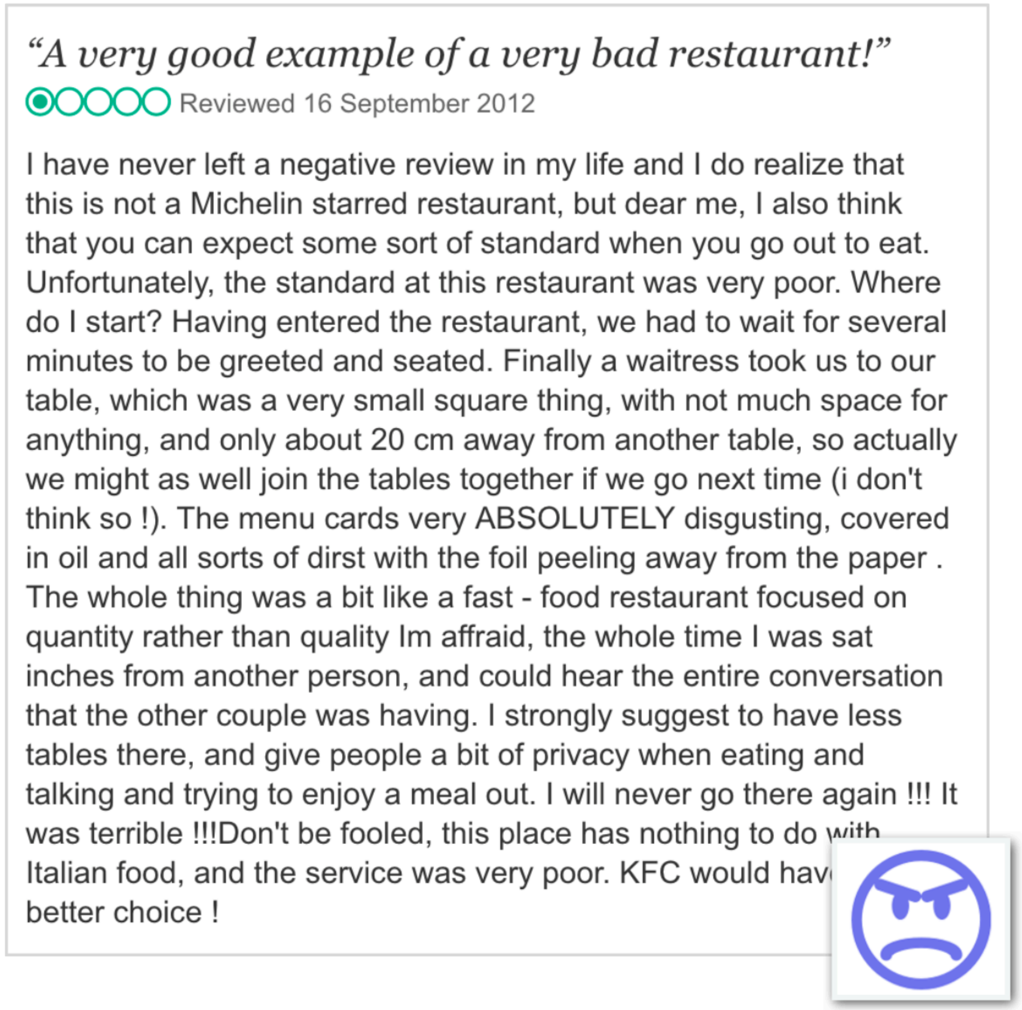
- **Example: Sentiment analysis** aims to analyse people's sentiments and opinions towards entities e.g. topics, products, individuals, organisations.
- The most fundamental task is classification of sentiment polarity:
  1. Build a binary classification model based on labelled training documents - i.e. examples of *positive* or *negative* text.
  2. Apply the model to new input documents to predict if they are *positive* or *negative*.



"Great Example of Good Hotel & Good Staff"

●●●●● Reviewed 4 May 2013 via mobile

Hotel was booked solid, but we got our reservation in just in time. The catch was we (2rooms) were just below the gym. Thankfully they were able to switch us to new rooms as guests checked out. Our new room was super comfortable with so much space. The breakfast was better than I remembered, with fresh berries and even granola. The staff was friendly and helpful throughout The restaurant was 5-star, amazing food. We will be back a soon. Thank you!



"A very good example of a very bad restaurant!"

●○○○○ Reviewed 16 September 2012

I have never left a negative review in my life and I do realize that this is not a Michelin starred restaurant, but dear me, I also think that you can expect some sort of standard when you go out to eat. Unfortunately, the standard at this restaurant was very poor. Where do I start? Having entered the restaurant, we had to wait for several minutes to be greeted and seated. Finally a waitress took us to our table, which was a very small square thing, with not much space for anything, and only about 20 cm away from another table, so actually we might as well join the tables together if we go next time (i don't think so !). The menu cards very ABSOLUTELY disgusting, covered in oil and all sorts of dirst with the foil peeling away from the paper . The whole thing was a bit like a fast - food restaurant focused on quantity rather than quality Im affraid, the whole time I was sat inches from another person, and could hear the entire conversation that the other couple was having. I strongly suggest to have less tables there, and give people a bit of privacy when eating and talking and trying to enjoy a meal out. I will never go there again !!! It was terrible !!!Don't be fooled, this place has nothing to do with Italian food, and the service was very poor. KFC would hav better choice !

Text Classification in Python

- Many of the classifiers provided by the Scikit-learn library are appropriate for text classification.
- We apply them in a similar way as with numeric data. However, we will need to preprocess documents beforehand.

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)
```

Preprocess training documents to build a document-term matrix *X*

```
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X, target)
```

Build a KNN model on the preprocessed training data

```
inputs = [doc]
input_X = vectorizer.transform(inputs)
```

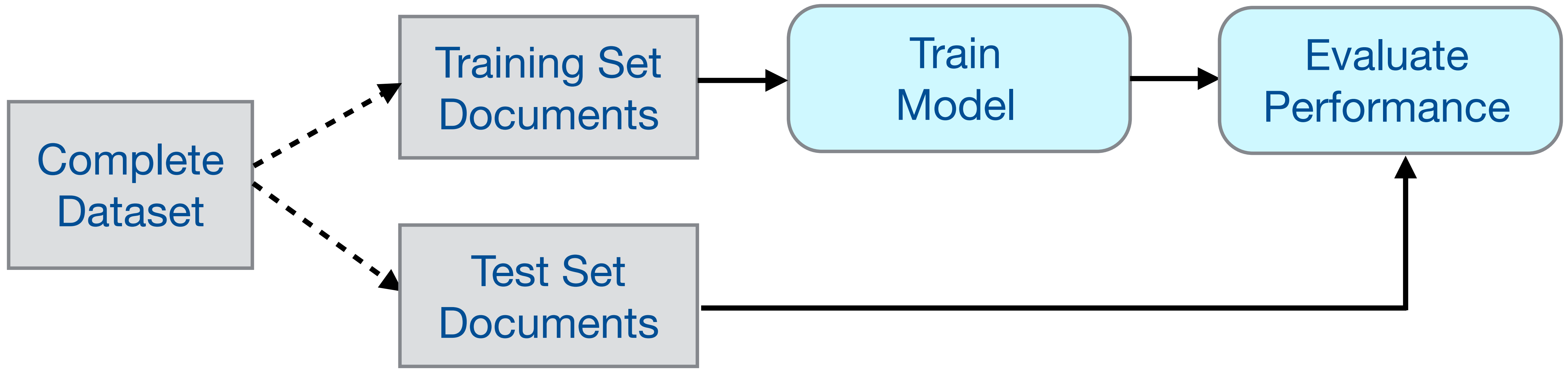
Preprocess new input document in the same way as the training data

```
model.predict(input_X)
```

Make a prediction for the class of the preprocessed input document

Evaluating Text Classifiers

- **Recall:** A key aspect of applying classification involves assessing how well an algorithm is performing at the task at hand.
- For evaluating text classifiers it is also common to split the complete corpus of annotated documents into two distinct sets:
  1. Training set: Set of documents provided to the classifier to build a model of the data. Each has been assigned a class label.
  2. Test set: Separate set of documents used to evaluate the classifier. The class labels are "hidden" from the algorithm.



Evaluating Text Classifiers in Python

- We can evaluate a KNN text classifier using a hold-out strategy:

```
train_docs, test_docs, train_target, test_target =
    train_test_split(docs, target, test_size=0.2)
```

Randomly split the original documents and target labels as 80% training and 20% test

```
vectorizer = CountVectorizer()
train_X = vectorizer.fit_transform(train_docs)
```

Preprocess the training set to build a document-term matrix

```
model = KNeighborsClassifier(n_neighbors=3)
model.fit(train_X, train_target)
```

Build a KNN model on the preprocessed training data

```
test_X = vectorizer.transform(test_docs)
```

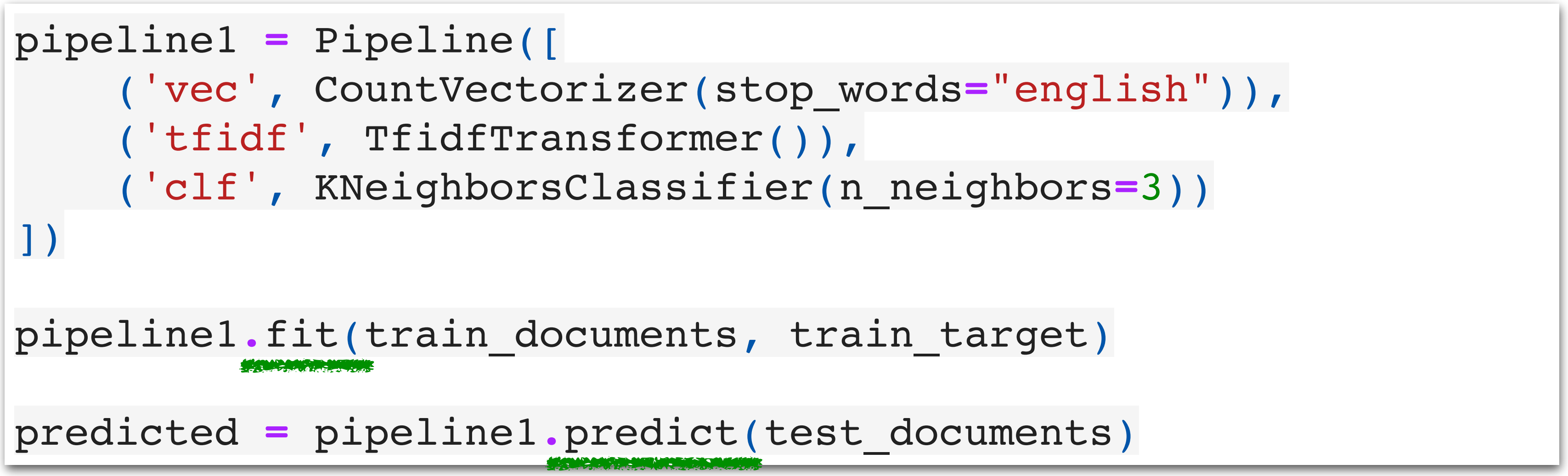
Create document-term matrix from test set. Call `transform()` to use the same vocabulary as before.

```
predicted = model.predict(test_X)
accuracy_score(test_target, predicted)
```

We can make predictions for our test set using the KNN model, and evaluate their accuracy

Scikit-learn Pipelines

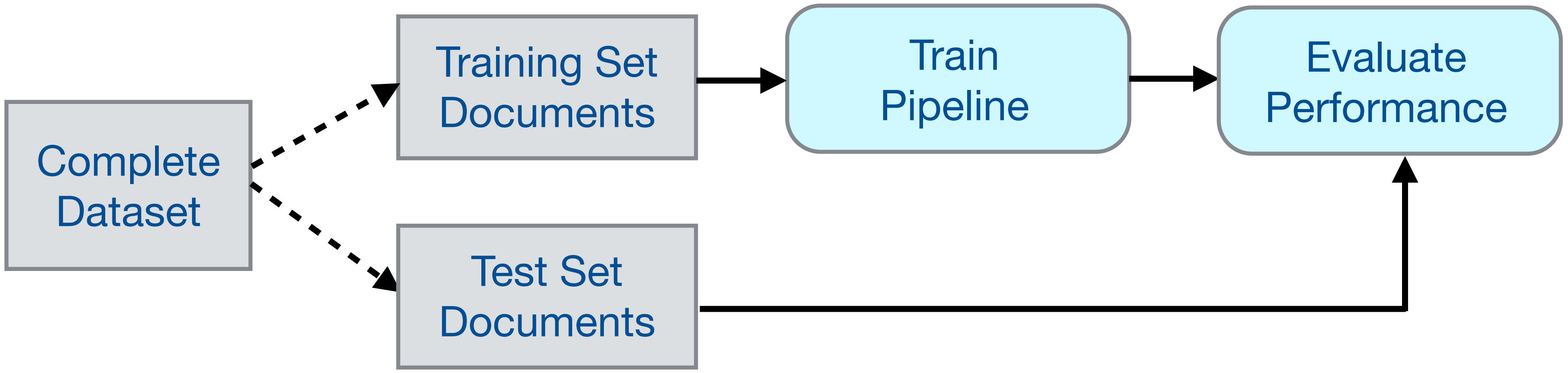
- Scikit-learn provides pipelines, which can be used to assemble several steps that can be evaluated together.
- A pipeline has associated `fit()` and `predict()` functions:



```
pipeline1 = Pipeline([
    ('vec', CountVectorizer(stop_words="english")),
    ('tfidf', TfidfTransformer()),
    ('clf', KNeighborsClassifier(n_neighbors=3))
])

pipeline1.fit(train_documents, train_target)

predicted = pipeline1.predict(test_documents)
```


Complete Dataset
Training Set Documents
Test Set Documents
Train Pipeline
Evaluate Performance