Lab: Text Processing and Machine Learning

Contents:

In this lab, we will be learning about Natural Language Processing (NLP), which can help computers analyse texts easily, for processing textual data for usage in machine learning models. NLP is a field in machine learning that allows a computer to understand, analyse, manipulate, and potentially generate human language. NLP has been used for applications such as: information retrieval, sentiment analysis, spam filter, and speech recognition by big tech companies.

The tool (library) we will be using today is NLTK (Natural Language Toolkit), which is a popular open-source package in Python. This package provides all common NLP tasks.

To install NLTK, in a Jupyter Notebook, type in the following code:

```
!pip install nltk
```

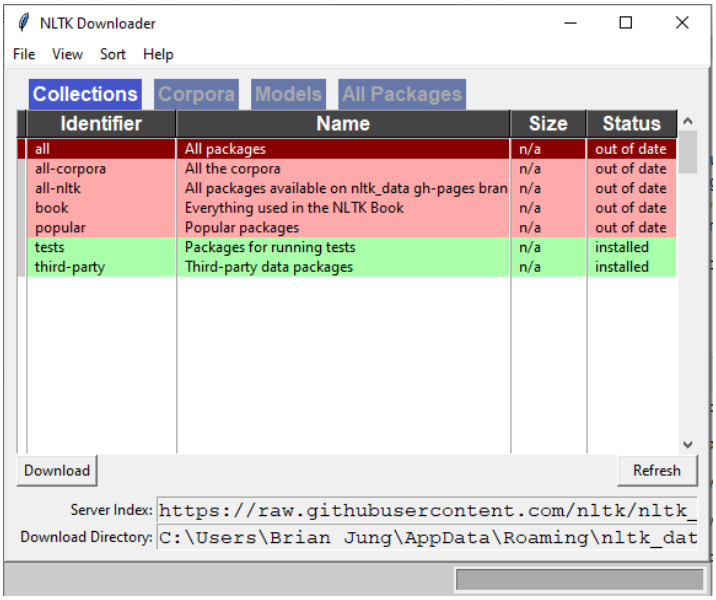
```
Requirement already satisfied: nltk in c:\users\brian jung\anaconda3\envs\python7\lib\site-packages
(3.5)
Requirement already satisfied: joblib in c:\users\brian jung\anaconda3\envs\python7\lib\site-packages
(from nltk) (0.17.0)
Requirement already satisfied: click in c:\users\brian jung\anaconda3\envs\python7\lib\site-packages
(from nltk) (7.1.2)
Requirement already satisfied: regex in c:\users\brian jung\anaconda3\envs\python7\lib\site-packages
(from nltk) (2021.3.17)
Requirement already satisfied: tqdm in c:\users\brian jung\anaconda3\envs\python7\lib\site-packages
(from nltk) (4.51.0)
```

In my machine, NLTK has already been installed, but yours should display different outputs if you have not downloaded the package yet. Let's try to import the library.

```
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

After running the above code, we get an NLTK Downloader Application which is helpful in downloading the necessary components of NLTK.



Right now, it will be helpful to download the following:

1. all-corpora
2. all-nltk
3. book
4. popular
5. tests
6. third-party

Select the Collections tab to see if these packages are installed. If they are not, go to All Packages tab and download them one by one.

1. Reading and exploring the dataset

We will be taking a look at the dataset "SMSSpamCollection.tsv". A .tsv file is similar to a .csv file, but it is tab separated instead of being comma separated. This data contains text messages on mobile phones and labels them according to whether they are spams or not. Spam messages are labelled as "spam" and messages that are not spam are labelled as "ham". Therefore, in this data, there are two columns: texts and their corresponding labels.

```
import pandas as pd

data = pd.read_csv('SMSSpamCollection.tsv', sep='\t', names=['label', 'body_text'], header=None)
data.head()
```

	label	body_text
0	ham	I've been searching for the right words to tha...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...
2	ham	Nah I don't think he goes to usf, he lives aro...
3	ham	Even my brother is not like to speak with me. ...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!

Data shape will output to be (5568, 2).

```
print('Out of {} rows, {} are spam, {} are ham'.format(len(data),
                                                       len(data[data['label'] == 'spam']),
                                                       len(data[data['label'] == 'ham'])))
```

```
Out of 5568 rows, 746 are spam, 4822 are ham
```

There are 746 spam texts and 4822 regular texts.

2. Preprocessing the data

Cleaning up the text data is necessary to highlight the attributes that we are going to want our machine learning models to pick up on. For text data, cleaning the data typically consists of a number of steps:

a) *Remove punctuation*

Punctuation can provide grammatical context to a sentence which supports our understanding. But for our vectoriser, which counts the number of words and not the context, it does not add value. Therefore, it is usually a good idea to remove all special characters.

For example, "How are you?" should be changed to "How are you".

We can see the list of punctuations available in the "string" package.

```
import string
string.punctuation
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
# function to remove punctuations
def remove_punct(text):
    text_nopunct = "".join([char for char in text if char not in string.punctuation])
    return text_nopunct
```

We will create a new column, "body_text_clean" that contains the texts that have no punctuations.

```
data['body_text_clean'] = data['body_text'].apply(lambda x: remove_punct(x))
data.head()
```

	label	body_text	body_text_clean
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL

As we can see, the punctuations in the raw texts have been deleted.

b) *Tokenisation*

Tokenisation separates texts into units such as sentences or words. It gives structure to previously unstructured text.

For example, "Plata o Plomo" becomes "'Plata','o','Plomo'".

To do this, we can simply use the split function in "re" package.

```
import re

# function to tokenise words
def tokenise(text):
    tokens = re.split('\W+', text) # W+ means that either a word character or a dash can go there.
    return tokens
```

We will again create a new column that contains the tokenised texts. We will apply the tokenisation on the cleaned text, and we will also convert all texts into lowercases as Python is case-sensitive.

```
data['body_text_tokenised'] = data['body_text_clean'].apply(lambda x: tokenise(x.lower()))
data.head()
```

	label	body_text	body_text_clean	body_text_tokenised
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[ive, been, searching, for, the, right, words,...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[i, have, a, date, on, sunday, with, will]

Here, we can see that each element of the new column contains a list of separated tokens, and they have been converted to lowercases.

c) *Remove stopwords*

Stopwords are common words that will likely appear in any text. They do not tell us much about our data, so it is a good practice to remove them for analysis. However, we must always be careful in removing stopwords as in some contexts, certain stopwords may be relevant to the context of the text.

For example, "silver or lead is fine for me" becomes "silver, lead, fine".

We will first create a list of stopwords taken from the NLTK package, which is located in the corpus package of NLTK.

```
import nltk

stopword = nltk.corpus.stopwords.words('english')

print(stopword[:10]) # example of stopwords

print(len(stopword))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
179
```

There are 179 stopwords identified and listed in the NLTK package.

```
# function to remove stopwords
def remove_stopwords(tokenised_list):
    text = [word for word in tokenised_list if word not in stopword]
    return text
```

We will create another column that contains the text without stopwords. We will apply this function over the tokenised text column.

```
data['body_text_nostop'] = data['body_text_tokenised'].apply(lambda x: remove_stopwords(x))
data.head()
```

	label	body_text	body_text_clean	body_text_tokenised	body_text_nostop
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[ive, been, searching, for, the, right, words,...	[ive, searching, right, words, thank, breather...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...	[free, entry, 2, wkly, comp, win, fa, cup, fin...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...	[nah, dont, think, goes, usf, lives, around, t...
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...	[even, brother, like, speak, treat, like, aids...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[i, have, a, date, on, sunday, with, will]	[date, sunday]

We can see that all unnecessary words, such as "been", "for", and "the", are removed.

d) *Preprocessing data: stemming*

Stemming helps to reduce a word to its stem form. Oftentimes, it makes sense to treat related words in the same way. It removes suffices, such as "ing", "ly", "s", etc. by a simple rule-based approach. Although this approach reduces the corpus of words, it often neglects the actual words. For example, "Entitling" or "Entitled" becomes "Entitl". In real-world search engines, they treat words with the same stem as synonyms.

While there are several stemming functions available, the most commonly used one is PorterStemmer, which is provided by NLTK. We will use this to convert the text data without stopwords to stemmed versions and put them into a separate column.

```
ps = nltk.PorterStemmer()

def stemming(tokenised_text):
    text = [ps.stem(word) for word in tokenised_text]
    return text

data['body_text_stemmed'] = data['body_text_nostop'].apply(lambda x: stemming(x))
data.head()
```

	label	body_text	body_text_clean	body_text_tokenised	body_text_nostop	body_text_stemmed
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[ive, been, searching, for, the, right, words,...	[ive, searching, right, words, thank, breather...	[ive, search, right, word, thank, breather, pr...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...	[free, entry, 2, wkly, comp, win, fa, cup, fin...	[free, entri, 2, wkli, comp, win, fa, cup, fin...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...	[nah, dont, think, goes, usf, lives, around, t...	[nah, dont, think, goe, usf, live, around, tho...
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...	[even, brother, like, speak, treat, like, aids...	[even, brother, like, speak, treat, like, aid,...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[i, have, a, date, on, sunday, with, will]	[date, sunday]	[date, sunday]

We can see that some words like "entry" and "wkly" are stemmed to "entri" and "wkli" even though they do not mean anything.

e) *Preprocessing data: Lemmatising*

Lemmatising derives the canonical form ('lemma') of a word, also known as the root form. It is better than stemming as it uses a dictionary-based approach, a morphological analysis to the root word. For example, "Entitling" and "Entitled" becomes "Entitle" rather than "Entitl" in stemming.

Stemming is typically faster as it simply shops off the end of the word without understanding the context of the word. Lemmatising is slower and more accurate as it takes an informed analysis with the context of the word in mind.

We can use WordNetLemmatizer from NLTK to perform this. We will apply this function to the texts without stopwords and put them in a new column.

```
wn = nltk.WordNetLemmatizer()

def lemmatising(tokenised_text):
    text = [wn.lemmatize(word) for word in tokenised_text]
    return text

data['body_text_lemmatised'] = data['body_text_nostop'].apply(lambda x: lemmatising(x))
data.head()
```

	label	body_text	body_text_clean	body_text_tokenised	body_text_nostop	body_text_stemmed	body_text_lemmatised
0	ham	I've been searching for the right words to tha...	Ive been searching for the right words to than...	[ive, been, searching, for, the, right, words,...	[ive, searching, right, words, thank, breather...	[ive, search, right, word, thank, breather, pr...	[ive, searching, right, word, thank, breather,...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...	[free, entry, 2, wkly, comp, win, fa, cup, fin...	[free, entri, 2, wkli, comp, win, fa, cup, fin...	[free, entry, 2, wkly, comp, win, fa, cup, fin...
2	ham	Nah I don't think he goes to usf, he lives aro...	Nah I dont think he goes to usf he lives aroun...	[nah, i, dont, think, he, goes, to, usf, he, l...	[nah, dont, think, goes, usf, lives, around, t...	[nah, dont, think, goe, usf, live, around, tho...	[nah, dont, think, go, usf, life, around, though]
3	ham	Even my brother is not like to speak with me. ...	Even my brother is not like to speak with me T...	[even, my, brother, is, not, like, to, speak, ...	[even, brother, like, speak, treat, like, aids...	[even, brother, like, speak, treat, like, aid,...	[even, brother, like, speak, treat, like, aid,...
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	I HAVE A DATE ON SUNDAY WITH WILL	[i, have, a, date, on, sunday, with, will]	[date, sunday]	[date, sunday]	[date, sunday]

For comparison, words like "chances" are lemmatised to "chance" whereas it was stemmed to "chanc".

For safety, we will save this data that we have preprocessed into a csv file.

```
data.to_csv('SMSSpamCollection_cleaned.csv', sep=',')
```

3. Vectorising

Vectorising is the process of encoding texts as integers, string to numeric form to create feature vectors so that machine learning models can understand our data.

a) *Vectorising: Bag-Of-Words*

Bag-of-Words (BoW) or CountVectoriser describes the presence of words within the text data. It gives a result of 1 if the word is present in the sentence and 0 if the word is not present. Therefore, it creates a bag of words witin a document-matrix count in each text document.

We will first read the data again and clean the raw data accordingly.

```
import pandas as pd
import re
import string
import nltk
pd.set_option('display.max_colwidth', 100) # to extend column width

stopwords = nltk.corpus.stopwords.words('english')
ps = nltk.PorterStemmer()

data = pd.read_csv('SMSSpamCollection.tsv', sep='\t')
data.columns = ['label', 'body_text']
```

We will remove the punctuations, tokenise the texts, remove stopwords, and stem the texts all in one-go to make it available for tokenisation.

```
def clean_text(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    tokens = re.split('\W+', text)
    text = [ps.stem(word) for word in tokens if word not in stopwords]
    return text
```

To create count vectoriser features, we can use CountVectorizer function in sklearn.feature_extraction.text. Here, we can also directly apply the "analyser" into CountVectorizer function to perform text cleaning before creating the vectors.

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer(analyzer=clean_text)
X_counts = count_vect.fit_transform(data['body_text'])

print(X_counts.shape)
print(count_vect.get_feature_names())
```

```
(5567, 8104)
['', '0', '008704050406', '0089mi', '0121', '01223585236', '01223585334', '0125698789', '02', '0206
03', '0207', '02070836089', '02072069400', '02073162414', '02085076972', '020903', '021', '050703',
'0578', '06', '060505', '061104', '07008009200', '07046744435', '07090201529', '07090298926', '0709
9833605', '071104', '07123456789', '0721072', '07732584351', '07734396839', '07742676969', '0775374
1225', '0776xxxxxxx', '07786200117', '077xxx', '078', '07801543489', '07808', '07808247860', '07808
726822', '07815296484', '07821230901', '0784987', '0789xxxxxxx', '079467462910788086786 7', '0796xxx
xxx', '07973788240', '07xxxxxxxxx', '0800', '08000407165', '08000776320', '08000839402', '080009307
05', '08000938767', '08001950382', '08002888812', '08002986030', '08002986906', '08002988890', '080
06344447', '0808', '08081263000', '08081560665', '0825', '0844', '08448350055', '08448714184', '084
5', '08450542832', '08452810071', '08452810073', '08452810075over18', '0870', '0870062117015 0p', '0
8701213186', '08701237397', '08701417012', '08701417012150p', '0870141701216', '087016248', '087017
52560', '087018728737', '0870241182716', '08702490080', '08702840625', '08702840625comuk', '0870443
9680', '08704439680tsc', '08706091795', '0870737910216yr', '08707500020', '08707509020', '087075333
1018', '08707808226', '08708034412', '08708800282', '08709222922', '08709501522', '0870k', '0871047
11148', '08712101358', '08712103738', '0871212025016', '08712300220', '087123002209am7pm', '0871231
7606', '08712400200', '08712400603', '08712402050', '08712402578', '08712402779', '08712402902', '0
8712402972', '08712404000', '08712405020', '08712405022', '08712460324', '08712460324nat', '0871246
6669', '0871277810710pmin', '0871277810810', '0871277810910pmin', '0871434239 92stop', '087147123779
```

We have fit the count vectoriser into the data that we have and have transformed the data accordingly. As we can see, there are more than 8000 columns now created, with each of them representing one particular word. As we can see here, each token may not be a word, but a number instead.

```
X_counts_df = pd.DataFrame(X_counts.toarray(), columns=count_vect.get_feature_names())
X_counts_df.head()
```

	0	008704050406	0089mi	0121	01223585236	01223585334	0125698789	02	020603	...	zindgi	zoe	zogtoriu	zoom	zo
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 8104 columns

We have arranged the converted vectorised data into a dataframe. As we can see, the data is quite sparse.

b) *Vectorising: N-Grams*

N-grams are simply all combinations of adjacent words or letters of length n that we can find in our source text. Ngrams with n=1 are called unigrams, n=2 are called bigrams, and n=3 are called trigrams.

For example, "plata o piomo means silver or lead" will be transformed into "['plata o', 'o plomo', 'plomo means', 'means silver', 'silver or', 'or lead']" for bigrams and "['plata o plomo', 'o plomo means', 'plomo means silver', 'means silver or', 'silver or lead']" for trigrams. It is simply grouping the words into n number of adjacent words.

Unigrams usually do not contain much information as compared to bigrams or trigrams. The basic principle behind n-grams is that they capture the letter or word that is likely to follow the given word. The longer the n-gram, te more context you have to work with.

We will apply n-grams onto the raw body text, but after cleaning the data. The n-gram can be applied directly into the CountVectorizer function. The count of each group word in a sentence word is stored in the document matrix.

```
from sklearn.feature_extraction.text import CountVectorizer

ngram_vect = CountVectorizer(ngram_range=(2, 2), analyzer=clean_text) # ngram_range=(2, 2) means bigrams, and (1, 1) means unigr
X_counts = ngram_vect.fit_transform(data['body_text'])

print(X_counts.shape)
print(ngram_vect.get_feature_names())
```

```
(5567, 8104)
['', '0', '008704050406', '0089mi', '0121', '01223585236', '01223585334', '0125698789', '02', '020603', '0207', '0207083608
9', '02072069400', '02073162414', '02085076972', '020903', '021', '050703', '0578', '06', '060505', '061104', '07008009200',
'07046744435', '07090201529', '07090298926', '07099833605', '071104', '07123456789', '0721072', '07732584351', '07734396839',
'07742676969', '07753741225', '0776xxxxxxx', '07786200117', '077xxx', '078', '07801543489', '07808', '07808247860', '07808726
822', '07815296484', '07821230901', '0784987', '0789xxxxxxx', '07946746291078808678 67', '0796xxxxxx', '07973788240', '07xxxxx
xxxx', '0800', '08000407165', '08000776320', '08000839402', '08000930705', '08000938767', '08001950382', '08002888812', '0800
2986030', '08002986906', '08002988890', '08006344447', '0808', '08081263000', '08081560665', '0825', '0844', '08448350055',
'08448714184', '0845', '08450542832', '08452810071', '08452810073', '08452810075over18', '0870', '08700621170150p', '08701213
186', '08701237397', '08701417012', '08701417012150p', '0870141701216', '087016248', '08701752560', '087018728737', '08702411
82716', '08702490080', '08702840625', '08702840625comuk', '08704439680', '08704439680tsc', '08706091795', '0870737910216yr',
'08707500020', '08707509020', '0870753331018', '08707808226', '08708034412', '08708800282', '08709222922', '08709501522', '08
70k', '087104711148', '08712101358', '08712103738', '0871212025016', '08712300220', '087123002209am7pm', '08712317606', '0871
2400200', '08712400603', '08712402050', '08712402578', '08712402779', '08712402902', '08712402972', '08712404000', '087124050
20', '08712405022', '08712460324', '08712460324nat', '08712466669', '0871277810710pmin', '0871277810810', '0871277810910pmi
n', '0871434239 92stop', '087147123779am7pm', '08714712379', '08714712388', '08714712394', '08714712412', '08714714011', '0871
4719523', '08715203028', '08715203649', '08715203652', '08715203656', '08715203677', '08715203685', '08715203694', '087152052
73', '08715500022', '08715705022', '08717111821', '08717168528', '08717205546', '08717507382', '08717507711', '08717509990',
'08717890890', '08717895698', '08717898035', '08718711108', '08718720201', '08718723815', '08718725756', '08718726270', '0871
```

We will save this into a dataframe as well.

```
X_counts_df = pd.DataFrame(X_counts.toarray(), columns=ngram_vect.get_feature_names())
X_counts_df.head()
```

	0	008704050406	0089mi	0121	01223585236	01223585334	0125698789	02	020603	...	zindgi	zoe	zogtoriu	zoom	zo
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 8104 columns

c) *Vectorising: TF-IDF*

TF-IDF computes the 'relative frequency' that a word appears in a document compared to its frequency across all documents. It is more useful than "term frequency" for identifying important words in each document. TF-IDF is computed by such:

TF-IDF

$$w_{i,j} = tf_{i,j} \times \log \frac{N}{df_i}$$

$tf_{i,j}$ = *number of times i occurs in j divided by total number of terms in j*
df_i = *number of documents containing i*
N = *total number of documents*

We will not go through the elaborate workings of TF-IDF, but it is enough to note that TF-IDF calculates some token's relative importance rather than its raw counts in the document.

Likewise, we will apply TF-IDF onto the body_text.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(analyzer=clean_text)
X_tfidf = tfidf_vect.fit_transform(data['body_text'])

print(X_tfidf.shape)
print(tfidf_vect.get_feature_names())
```

```
'08715203028', '08715203649', '08715203652', '08715203656', '08715203677', '08715203685', '08715203
694', '08715205273', '08715500022', '08715705022', '08717111821', '08717168528', '08717205546', '08
717507382', '08717507711', '08717509990', '08717890890', '08717895698', '08717898035', '0871871110
8', '08718720201', '08718723815', '08718725756', '08718726270', '08718726270150gbpmtmsg18', '087187
26970', '08718726971', '08718726978', '087187272008', '08718727868', '08718727870', '08718729755',
'08718729758', '08718730555', '08718730666', '08718738001', '08718738002', '08718738034', '08719180
219', '08719180248', '08719181259', '08719181503', '08719181513', '08719839835', '08719899217', '08
719899229', '08719899230', '09041940223', '09050000301', '09050000332', '09050000460', '0905000055
5', '09050000878', '09050000928', '09050001295', '09050001808', '09050002311', '09050003091', '0905
0005321', '09050090044', '09050280520', '09053750005', '09056242159', '09057039994', '09058091854',
'09058091870', '09058094454', '09058094455', '09058094507', '09058094565', '09058094583', '09058094
594', '09058094597', '09058094599', '09058095107', '09058095201', '09058097189', '09058097218', '09
058098002', '09058099801', '09061104276', '09061104283', '09061209465', '09061213237', '0906122106
1', '09061221066', '09061701444', '09061701461', '09061701851', '09061701939', '09061702893', '0906
1743386', '09061743806', '09061743810', '09061743811', '09061744553', '09061749602', '09061790121',
'09061790125', '09061790126', '09063440451', '09063442151', '09063458130', '0906346330', '090640110
00', '09064012103', '09064012160', '09064015307', '09064017295', '09064017305', '09064018838', '090
64019014', '09064019788', '09065069120', '09065069154', '09065171142stopsms08', '09065171142stopsms
08718727870150ppm', '09065174042', '09065394514', '09065394973', '09065989180', '09065989182', '090
66350750', '09066358152', '09066358361', '09066361921', '09066362206', '09066362220', '0906636223
```

```
X_tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=tfidf_vect.get_feature_names())
X_tfidf_df.head()
```

	0	008704050406	0089mi	0121	01223585236	01223585334	0125698789	02	020603	...	zindgi	zoe	zogtoriu	zoom
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
1	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	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
3	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
4	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

5 rows × 8104 columns

As usual, this results in a sparse matrix, which contains mostly 0 for most entries.

4. Feature engineering: feature creation

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning models work better. It is akin to an art as it requires domain knowledge and it can be tough to create features, but it can be fruitful for machine learning models to predict results as they can be related heavily to the prediction.

For this case, it may be useful to get the length of text messages and the percentage of punctuations in text messages. Spam messages may be longer and may contain more punctuations than normal messages.

```
data.head()
```

	label	body_text
0	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ...
1	ham	Nah I don't think he goes to usf, he lives around here though
2	ham	Even my brother is not like to speak with me. They treat me like aids patent.
3	ham	I HAVE A DATE ON SUNDAY WITH WILL!!
4	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your call...

```
import string

# function to calculate length of messages excluding space
data['body_len'] = data['body_text'].apply(lambda x: len(x) - x.count(' '))
data.head()
```

	label	body_text	body_len
0	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ...	128
1	ham	Nah I don't think he goes to usf, he lives around here though	49
2	ham	Even my brother is not like to speak with me. They treat me like aids patent.	62
3	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	28
4	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your call...	135

```
def count_punct(text):
    count = sum([1 for char in text if char in string.punctuation])
    return round(count/(len(text) - text.count(' ')), 3) * 100

data['punct%'] = data['body_text'].apply(lambda x: count_punct(x))
data.head()
```

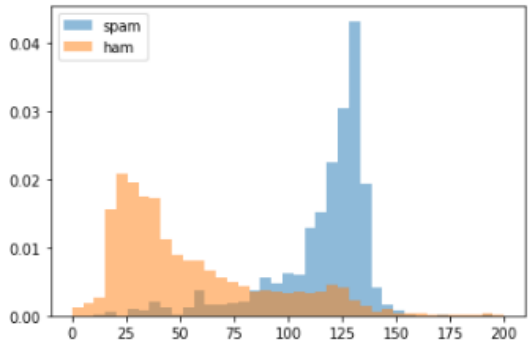
	label	body_text	body_len	punct%
0	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ...	128	4.7
1	ham	Nah I don't think he goes to usf, he lives around here though	49	4.1
2	ham	Even my brother is not like to speak with me. They treat me like aids patent.	62	3.2
3	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	28	7.1
4	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your call...	135	4.4

a) *Checking if the engineered features are good or not*

We will perform some visualisation to determine if the features created are good features.

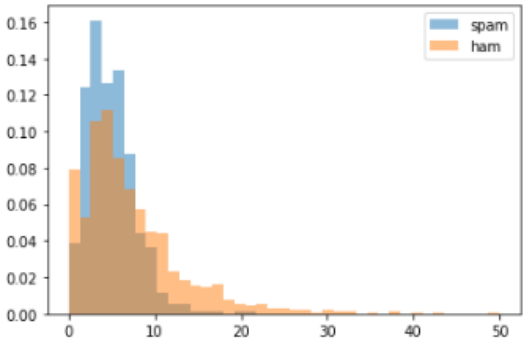
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

```
bins = np.linspace(0, 200, 40)

plt.hist(data[data['label']=='spam']['body_len'], bins, alpha=0.5, density=True, stacked=True, label='spam')
plt.hist(data[data['label']=='ham']['body_len'], bins, alpha=0.5, density=True, stacked=True, label='ham')
plt.legend(loc='upper left')
plt.show()
```



We can clearly see that spams have a high number of words as compared to hams. So it seems to be a good feature to use.

```
bins = np.linspace(0, 50, 40)

plt.hist(data[data['label'] == 'spam']['punct%'], bins, alpha=0.5, density=True, stacked=True, label='spam')
plt.hist(data[data['label'] == 'ham']['punct%'], bins, alpha=0.5, density=True, stacked=True, label='ham')
plt.legend(loc='upper right')
plt.show()
```



Spam has a percentage of punctuations but not that far away from ham. This is surprising as there are times where spam emails can contain a lot of punctuation marks. However, in this case, it can still be identified as a good feature to use.

5. Building ML classifiers: Model Selection

Now, we have to classify whether the texts are spams or hams. We can use an ensemble method of machine learning, where multiple models are used, and their combination produces better results than a single model (SVM/Naive Bayes). Ensemble methods have been proven to be quite accurate, also being the first-choice model for many Kaggle competitions.

Random Forest (multiple random decision trees) are constructed, and the aggregates of each tree are used for the final prediction in this case.

We will also use the following methods:

1. Grid-search: it exhaustively searches overall parameter combinations in a given grid to determine the best hyperparameters for the model.
2. Cross-validation: it divides the data into k subsets and repeats the method k times where a different subset is used as the test set in each iteration.

We will create two different types of data - one with features created using count vectoriser and another with features created using TF-IDF. The vectorised data will then be combined with the engineered features early on. The resulting two dataframes will be named "X_tfidf_feat" and "X_count_feat".

```
import warnings
warnings.filterwarnings('ignore', category=DeprecationWarning)
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# TF-IDF data
tfidf_vect = TfidfVectorizer(analyzer=clean_text)
X_tfidf = tfidf_vect.fit_transform(data['body_text'])
X_tfidf_feat = pd.concat([data['body_len'], data['punct%'], pd.DataFrame(X_tfidf.toarray())], axis=1)

# Count Vectoriser data
count_vect = CountVectorizer(analyzer=clean_text)
X_count = count_vect.fit_transform(data['body_text'])
X_count_feat = pd.concat([data['body_len'], data['punct%'], pd.DataFrame(X_count.toarray())], axis=1)
```

```
X_tfidf_feat.head()
```

	body_len	punct%	0	1	2	3	4	5	6	7	...	8094	8095	8096	8097	8098	8099	8100	8101	8102	8103
0	128	4.7	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	0.0	0.0	0.0	0.0
1	49	4.1	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	0.0	0.0	0.0	0.0
2	62	3.2	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	0.0	0.0	0.0	0.0
3	28	7.1	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	0.0	0.0	0.0	0.0
4	135	4.4	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	0.0	0.0	0.0	0.0

5 rows × 8106 columns

```
X_count_feat.head()
```

	body_len	punct%	0	1	2	3	4	5	6	7	...	8094	8095	8096	8097	8098	8099	8100	8101	8102	8103
0	128	4.7	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	49	4.1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	62	3.2	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	28	7.1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	135	4.4	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 8106 columns

After the data has been organised and prepared, we will run Random Forest classifier, with some hyperparameter grids and output the results. The results will be organised into a dataframe, highlighting various results. The target (labels) for the data will be taken from data['label'] variable.

a) *Count Vectoriser classification*

```
rf = RandomForestClassifier()
param = {'n_estimators': [10, 150, 300],
        'max_depth': [30, 60, 90, None]}

gs = GridSearchCV(rf, param, cv=5, n_jobs=4) # n_jobs for parallelising search
gs_fit = gs.fit(X_count_feat, data['label'])
pd.DataFrame(gs_fit.cv_results_).sort_values('mean_test_score', ascending=False).head()
```

param_max_depth	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score
90	10	{'max_depth': 90, 'n_estimators': 10}	0.974865	0.980251	0.971249	0.968553	0.973944	0.973773
None	300	{'max_depth': None, 'n_estimators': 300}	0.978456	0.975763	0.973046	0.966757	0.970350	0.972874
90	150	{'max_depth': 90, 'n_estimators': 150}	0.980251	0.973968	0.972147	0.966757	0.969452	0.972515
90	300	{'max_depth': 90, 'n_estimators': 300}	0.979354	0.972172	0.974843	0.967655	0.967655	0.972336
None	150	{'max_depth': None, 'n_estimators': 150}	0.977558	0.975763	0.972147	0.967655	0.968553	0.972335

The mean test score, which is represented in accuracy (%), is the highest when n_estimators is 10 and max_depth is 90. n_estimators is the number of trees in the random forest (group of decision trees) and max_depth is the maximum number of levels in each decision tree. The average accuracy, represented in mean_test_score, is 0.973773, close to 97.38%.

b) *TF-IDF Classification*

```
rf = RandomForestClassifier()
param = {'n_estimators': [10, 150, 300],
        'max_depth': [30, 60, 90, None]}

gs = GridSearchCV(rf, param, cv=5, n_jobs=4)
gs_fit = gs.fit(X_tfidf_feat, data['label'])
pd.DataFrame(gs_fit.cv_results_).sort_values('mean_test_score', ascending=False).head()
```

param_max_depth	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score
None	150	{'max_depth': None, 'n_estimators': 150}	0.977558	0.977558	0.974843	0.967655	0.973046	0.974132
90	150	{'max_depth': 90, 'n_estimators': 150}	0.976661	0.979354	0.973944	0.966757	0.971249	0.973593
90	300	{'max_depth': 90, 'n_estimators': 300}	0.976661	0.973968	0.974843	0.969452	0.972147	0.973414
60	300	{'max_depth': 60, 'n_estimators': 300}	0.977558	0.974865	0.973046	0.967655	0.970350	0.972695
None	300	{'max_depth': None, 'n_estimators': 300}	0.977558	0.976661	0.972147	0.966757	0.969452	0.972515

For TF-IDF classification, the highest scored n_estimators is 150 and max_depth is none. The highest mean_test_score is 0.974132, which is close to 97.41% accuracy. It did marginally better than count vectoriser, but this improvement is not significant enough.

6. Conclusion

There are numerous methods in cleaning, preprocessing, and preparing textual data for use in machine learning applications. Here, we have touched on a few of those methods. There are more complex methods available for handling textual data, which you can explore on your own. For this task of classifying spam messages, the accuracy was high since the problem was not extremely complicated. However, more analysis of the results can be done using the various methods discussed earlier.

7. Assignment

There will be another data provided to you, "movie_data.csv". This data contains reviews and their corresponding sentiments. The reviews are online movie reviews. There are 2 classes of sentiments: 1 means positive and 0 means negative sentiment. Your task is to classify whether the movie reviews are positive or negative by preparing the textual data provided in the reviews.

Using the data "movie_data.csv" that has been provided to you, perform the following activities:

1. Preprocess the data according to what has been shown in this guide.
2. Vectorise the data according to what has been shown in this guide.
3. Using SVM, Gradient Boosting trees, and Random Forest, perform classification
4. Compare the results for the machine learning models and discuss the reasons.

```
df = pd.read_csv('movie_data.csv')
df.head()
```

	review	sentiment
0	I went and saw this movie last night after being coaxed to by a few friends of mine. I'll admit ...	1
1	Actor turned director Bill Paxton follows up his promising debut, the Gothic-horror "Frailty", w...	1
2	As a recreational golfer with some knowledge of the sport's history, I was pleased with Disney's...	1
3	I saw this film in a sneak preview, and it is delightful. The cinematography is unusually creati...	1
4	Bill Paxton has taken the true story of the 1913 US golf open and made a film that is about much...	1

```
df.shape
```

```
(50000, 2)
```

An example of the data is shown above, with 50,000 reviews included in the data.