BUILDING THE SENTIMENT ANALYSIS SOLUTION BY SELECTING AN APPROPRIATE DATASET AND PRE-PROCESSING THE DATA

TEAM MEMBER :

810021106011 : ARJUNBABU R S

PHASE 3 : PROJECT SUBMISSION

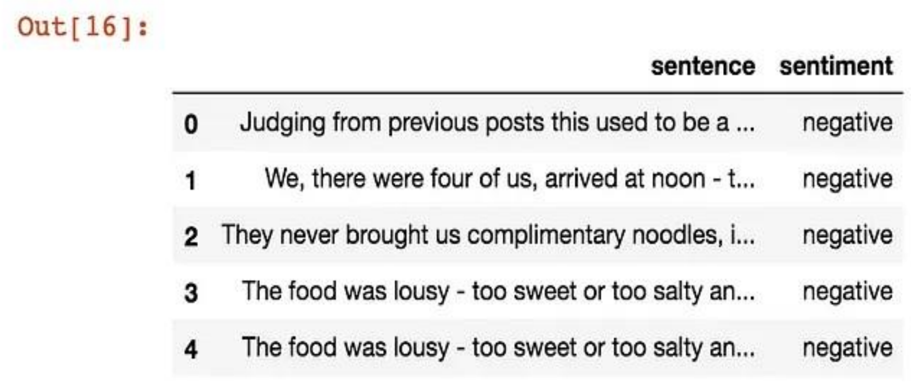
Sentiment analysis of reviews: Text Pre-processing

Training Data

To make a model you first need training data. I was able to find labeled training data for sentiment evaluation of restaurant reviews in New York from meta-share, a language data resource. It can be found here and can be freely downloaded as an xml file when signing up.

Lets first look at the data. I only extracting the data that is necessary for my model. The first column showing the polarity of the review and second column text of the review.

```
Import pandas as pd
Import numpy as np
Import xml.etree.ElementTree as ET
Xml_path = './NLP/ABSA15_RestaurantsTrain2/ABSA-15_Restaurants_Train_Final.xml'
Def parse_data_2015(xml_path):
  Container = []
  Reviews = ET.parse(xml_path).getroot()

  For review in reviews:
    Sentences = review.getchildren()[0].getchildren()
    For sentence in sentences:
      Sentence_text = sentence.getchildren()[0].text

      Try:
        Opinions = sentence.getchildren()[1].getchildren()

        For opinion in opinions:
          Polarity = opinion.attrib[“polarity”]
          Target = opinion.attrib[“target”]

          Row = {“sentence”: sentence_text, “sentiment”:polarity}
          Container.append(row)

      Except IndexError:
        Row = {“sentence”: sentence_text}
        Container.append(row)

  Return pd.DataFrame(container)
ABSA_df = parse_data_2015(xml_path)
ABSA_df.head()
```

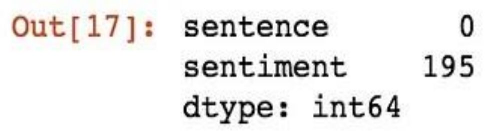


```
Out[16]:
```

	sentence	sentiment
0	Judging from previous posts this used to be a ...	negative
1	We, there were four of us, arrived at noon - t...	negative
2	They never brought us complimentary noodles, i...	negative
3	The food was lousy - too sweet or too salty an...	negative
4	The food was lousy - too sweet or too salty an...	negative

Before we start cleaning the text I wanted to ensure duplicates were dropped. Since we are using sentiment as our target (the variable we will be predicting) we cannot have any null values so these are dropped leaving the us with 1,201 rows.

ABSA_df.isnull().sum()



```
Out[17]: sentence       0
         sentiment    195
         dtype: int64
```

Print “Original:”, ABSA_df.shape

ABSA_dd = ABSA_df.drop_duplicates()

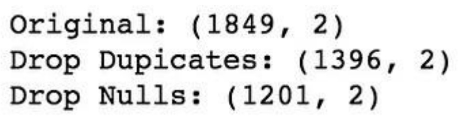
Dd = ABSA_dd.reset_index(drop=True)

Print “Drop Dupicates:”, dd.shape

Dd_dn = dd.dropna()

Df = dd_dn.reset_index(drop=True)

Print “Drop Nulls:”, df.shape



```
Original: (1849, 2)
Drop Dupicates: (1396, 2)
Drop Nulls: (1201, 2)
```

Dirty dirty text

Of all data, text is the most unstructured form and so means we have a lot of cleaning to do. These pre-processing steps help convert noise from high dimensional features to the low dimensional space to obtain as much accurate information as possible from the text.

Preprocessing data can consist of many steps depending on the data and the situation. To guide me through cleaning, I used a blogpost from analytics vidhya which shows the process of cleaning tweets. As we are dealing with reviews, some of the methods will not apply here.

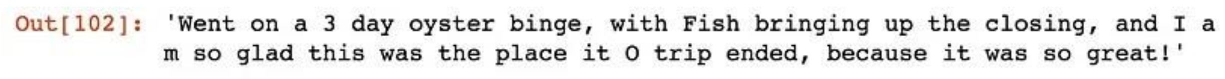
To further organise this process a blogpost from kdnuggets split it into categories of tokenization, normalization and substitution.

Preprocessing:

Tokenization

Tokenization is the process of converting text into tokens before transforming it into vectors. It is also easier to filter out unnecessary tokens. For example, a document into paragraphs or sentences into words. In this case we are tokenising the reviews into words.

Df.sentence[17]

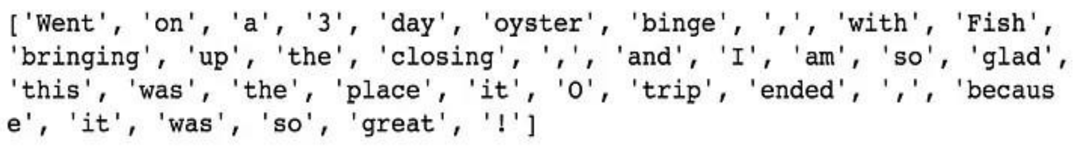


```
Out[102]: 'Went on a 3 day oyster binge, with Fish bringing up the closing, and I a
          m so glad this was the place it O trip ended, because it was so great!'
```

From nltk.tokenize import word_tokenize

Tokens = word_tokenize(df.sentence[17])

Print(tokens)



```
['Went', 'on', 'a', '3', 'day', 'oyster', 'binge', ',', 'with', 'Fish',
'bringing', 'up', 'the', 'closing', ',', 'and', 'I', 'am', 'so', 'glad',
'this', 'was', 'the', 'place', 'it', 'O', 'trip', 'ended', ',', 'becaus
e', 'it', 'was', 'so', 'great', '!']
```

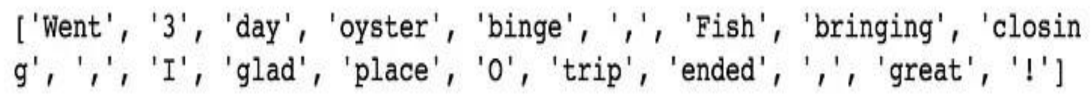
Stopwords

Stop words are the most commonly occuring words which are not relevant in the context of the data and do not contribute any deeper meaning to the phrase. In this case contain no sentiment. NLTK provide a library used for this.

From nltk.corpus import stopwords

Stop_words = stopwords.words('english')

Print [I for I in tokens if I not in stop_words]



```
['Went', '3', 'day', 'oyster', 'binge', ',', 'Fish', 'bringing', 'closin
g', ',', 'I', 'glad', 'place', 'O', 'trip', 'ended', ',', 'great', '!']
```

Preprocessing:

Normalization

Words which look different due to casing or written another way but are the same in meaning need to be process correctly. Normalisation processes ensure that these words are treated equally. For example, changing numbers to their word equivalents or converting the casing of all the text.

‘100’ → ‘one hundred’

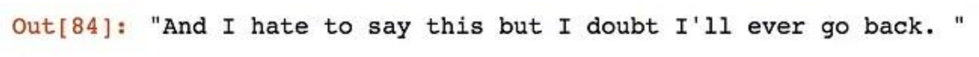
‘Apple’ → ‘apple’

The following normalisation changes are made:

1. **Casing the Characters**

Converting character to the same case so the same words are recognised as the same. In this case we converted to lowercase.

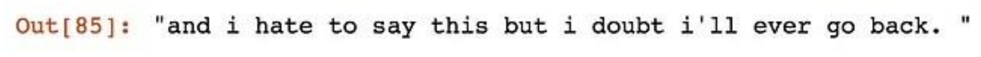
Df.sentence[24]



```
Out[84]: "And I hate to say this but I doubt I'll ever go back. "
```

Lower_case = df.sentence[24].lower()

Lower_case

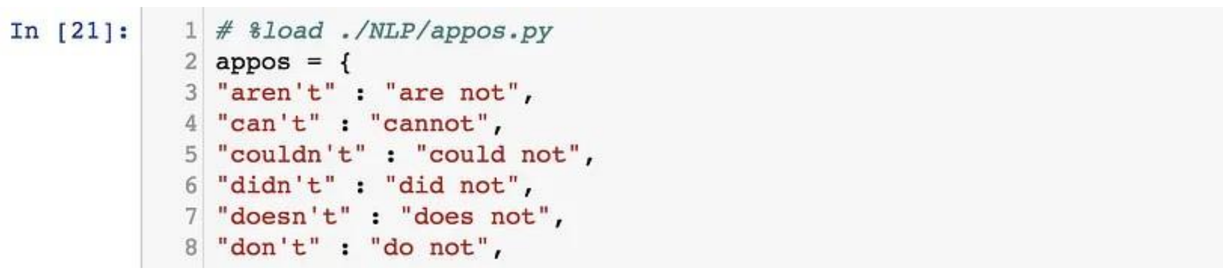


```
Out[85]: "and i hate to say this but i doubt i'll ever go back. "
```

2. Negation handling

Apostrophes connecting words are used everywhere, especially in public reviews. To maintain uniform structure it is recommended they should be converted into standard lexicons. The text will then follow the rules of context free grammar and helps avoids any word-sense disambiguation.

There was an apostrophe dictionary found from user comments on analytics vidhy blog which you can download here. The dictionary is in lowercase so the conversion will follow from the lower casing above.



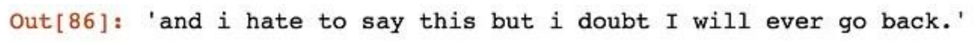
```
In [21]:  # %load ./NLP/appos.py
          appos = {
          "aren't" : "are not",
          "can't" : "cannot",
          "couldn't" : "could not",
          "didn't" : "did not",
          "doesn't" : "does not",
          "don't" : "do not",
```

Words = lower_case.split()

Reformed = [appos[word] if word in appos else word for word in words]

Reformed = “ “.join(reformed)

Reformed

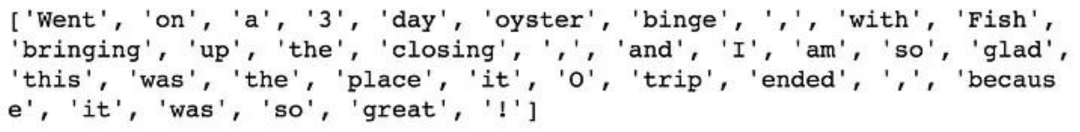


```
Out[86]:  'and i hate to say this but i doubt I will ever go back.'
```

3 . Removing

Stand alone punctuations, special characters and numerical tokens are removed as they do not contribute to sentiment which leaves only alphabetic characters. This step needs the use of tokenized words as they have been split appropriately for us to remove.

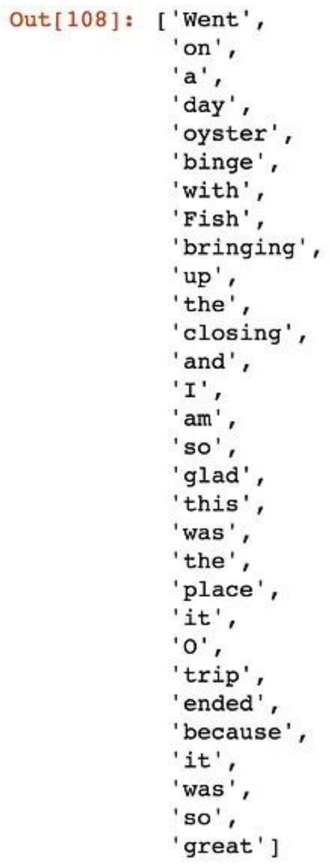
Tokens



```
['Went', 'on', 'a', '3', 'day', 'oyster', 'binge', ',', 'with', 'Fish',
'bringing', 'up', 'the', 'closing', ',', 'and', 'I', 'am', 'so', 'glad',
'this', 'was', 'the', 'place', 'it', 'O', 'trip', 'ended', ',', 'becaus
e', 'it', 'was', 'so', 'great', '!']
```

Words = [word for word in tokens if word.isalpha()]

Words



```
Out[108]: ['Went',
           'on',
           'a',
           'day',
           'oyster',
           'binge',
           'with',
           'Fish',
           'bringing',
           'up',
           'the',
           'closing',
           'and',
           'I',
           'am',
           'so',
           'glad',
           'this',
           'was',
           'the',
           'place',
           'it',
           'O',
           'trip',
           'ended',
           'because',
           'it',
           'was',
           'so',
           'great']
```

4.**Lemmatization**

This process finds the base or dictionary form of the word known as the lemma. This is done through the use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations). This normalization is similar to stemming but takes into account the context of the word.

‘are’, ‘is’, ‘being’ → ‘be’

Gensim: Lemmatization

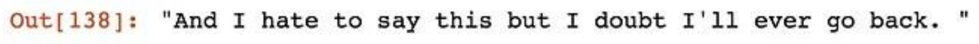
I used the lemmatize method from the Gensim package which took care of lower casing, removal of numerics, stand alone punctuation, special characters and stop words. It also identifies the words with part-of-speech tagging (POS- tagging) considering nouns/ NN, verbs/VB, adjectives/JJ and adverbs/RB.

The function uses the English lemmatizer from the pattern library to extract their lemmas. The words in a text are identified through word-category disambiguation where both its definition and context are taken into account to identify the specific POS- tag.

Tip1: This function is only available when the optional ‘pattern’ package is installed!

Tip2: This function only applies to UTF-8 encoded tokens.

Df.sentence[24]

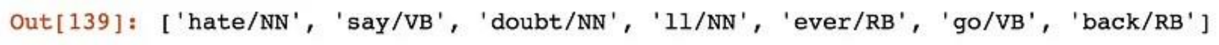


```
Out[138]: "And I hate to say this but I doubt I'll ever go back. "
```

From gensim.utils import lemmatize

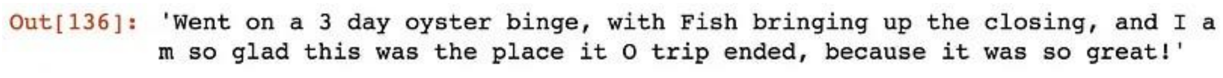
Lemm = lemmatize(df.sentence[24])

Lemm



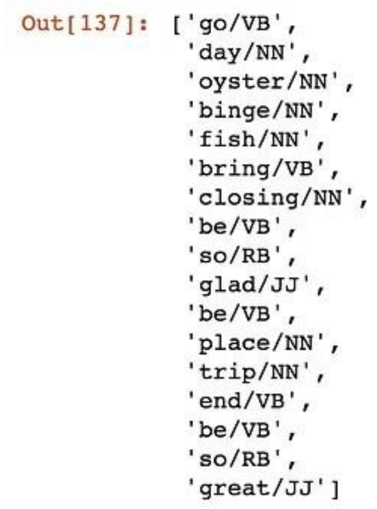
```
Out[139]: ['hate/NN', 'say/VB', 'doubt/NN', 'll/NN', 'ever/RB', 'go/VB', 'back/RB']
```

Df.sentence[17]



```
Out[136]: 'Went on a 3 day oyster binge, with Fish bringing up the closing, and I a
          m so glad this was the place it O trip ended, because it was so great!'
```

Lemmatize(df.sentence[17])



```
Out[137]: ['go/VB',
           'day/NN',
           'oyster/NN',
           'binge/NN',
           'fish/NN',
           'bring/VB',
           'closing/NN',
           'be/VB',
           'so/RB',
           'glad/JJ',
           'be/VB',
           'place/NN',
           'trip/NN',
           'end/VB',
           'be/VB',
           'so/RB',
           'great/JJ']
```

Preprocessing:

Substitution

This involves removing noise from text in its raw format. For example, the text is scrapped from the web it may contain HTML or XML wrappers or markups. Removal of these can be done through regular expressions. Fortunately our reviews do not apply to this as we were able to extract the exact review from the XML file.

Decoding

I found it difficult investigate what decoding means in the the NLP context however, I came across a comment from a stackoverflow question which helped me break down this section.

Text data is subject to different formats of decoding. For example, ASCII which contains english based letters, control characters, punctuation and numbers. When dealing with other languages with non-Latin characters another format may need to apply.

Unicode contains character sets including all languages by assigning every character to a unique number. It is recommended to use UTF-8. With its 1–4 bytes per character memory it widely accepts the majority of languages in the first 2 bytes and is memory efficient if dealing with mostly ASCII characters.

UTF-8 Character Bytes

1 byte: Standard ASCII

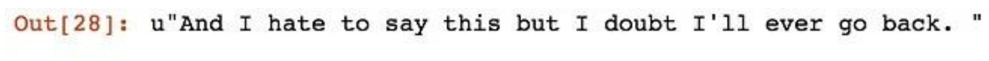
2 bytes: Arabic, Hebrew, most European scripts

3 bytes: BMP

4 bytes: All Unicode characters.

As we could be working with text it is advised to decode the utf-8 format ensuring they are all in the same format. The output presents a “u” before the string indicating it is Unicode.

```
Df.sentence[24].decode(“utf-8-sig”)
```



```
Out[28]: u"And I hate to say this but I doubt I'll ever go back. "
```

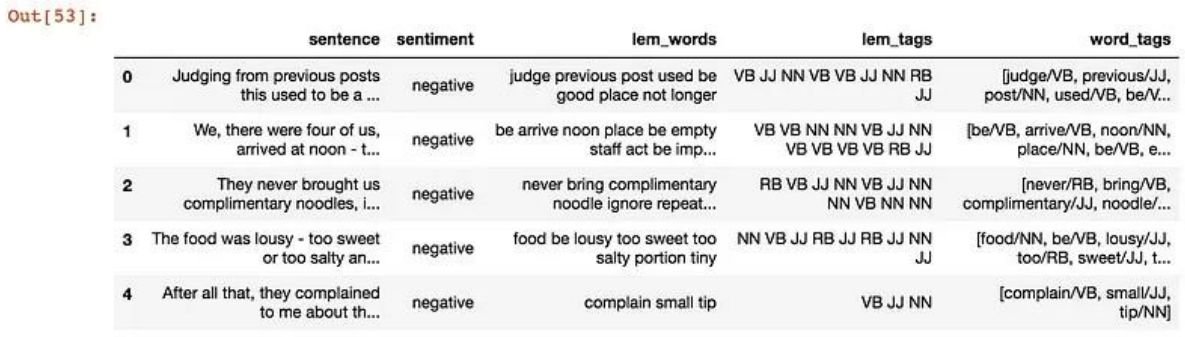
Defining a Cleaning Function

I created a cleaning function which will be applied to the whole dataset. It includes decoding, lowercasing (so the negation dictionary can apply), conversion with negation dictionary and lemmatization which includes lower casing, tokenising, removal of special characters, stand alone punctuation, stop words and POS tagging. The second function is to separate the tags and words.

```
Def cleaning_function(tips):
   All_ = []
   For tip in tqdm(tips):
      Time.sleep(0.0001)

#      Decoding function
      Decode = tip.decode(“utf-8-sig”)

#      Lowercasing before negation
      Lower_case = decode.lower()
```

```
#       Replace apostrophes with words

        Words = lower_case.split()

        Split = [appos[word] if word in appos else word for word in words]

        Reformed = " ".join(split)


#       Lemmatization

        Lemm = lemmatize(lower_case)

        All_.append(lemm)


    Return all_

Def separate_word_tag(df_lem_test):

    Words=[]

    Types=[]

    Df= pd.DataFrame()

    For row in df_lem_test:

        Sent = []

        Type_ =[]

        For word in row:

            Split = word.split('/')

            Sent.append(split[0])

            Type_.append(split[1])

Words.append(' '.join(word for word in sent))

        Types.append(' '.join(word for word in type_))

Df['lem_words']= words

    Df['lem_tag']= types

    Return df

Cleaning Training Data

Word_tag = cleaning_function(df.sentence)
```

```
Lemm_df = separate_word_tag(word_tag)

# concat cleaned text with original

Df_training = pd.concat([df, lemm_df], axis=1)

Df_training['word_tags'] = word_tag

Df_training.head()
```



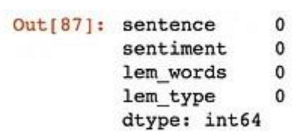
Out[53]:

	sentence	sentiment	lem_words	lem_tags	word_tags
0	Judging from previous posts this used to be a ...	negative	judge previous post used be good place not longer	VB JJ NN VB VB JJ NN RB JJ	[judge/VB, previous/JJ, post/NN, used/VB, be/V...
1	We, there were four of us, arrived at noon - t...	negative	be arrive noon place be empty staff act be imp...	VB VB NN NN VB JJ NN VB VB VB VB RB JJ	[be/VB, arrive/VB, noon/NN, place/NN, be/VB, e...
2	They never brought us complimentary noodles, i...	negative	never bring complimentary noodle ignore repeat...	RB VB JJ NN VB JJ NN NN VB NN NN	[never/RB, bring/VB, complimentary/JJ, noodle/...
3	The food was lousy - too sweet or too salty an...	negative	food be lousy too sweet too salty portion tiny	NN VB JJ RB JJ RB JJ NN JJ	[food/NN, be/VB, lousy/JJ, too/RB, sweet/JJ, t...
4	After all that, they complained to me about th...	negative	complain small tip	VB JJ NN	[complain/VB, small/JJ, tip/NN]

Check for null and empty values

There were no null values found because the cleaning process does not input nulls into empty values. Both were checked and the empty values were removed. It was identified that the text of the review was 3 reviews including “10”, “LOL” and “Why?” which has changed into null values during the cleaning process. I felt they did not contribute to the sentiment analysis so thought removing them is valid.

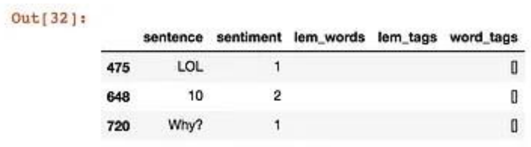
```
# reset index just to be safe

Df_training = df_training.reset_index(drop=True)

#check null values

Df_training.isnull().sum()
```



```
Out[87]: sentence     0
         sentiment    0
         lem_words    0
         lem_type     0
         dtype: int64
```

# empty values

Df_training[df_training['lem_words']=='']



Out[32]:

	sentence	sentiment	lem_words	lem_tags	word_tags
475	LOL	1			[]
648	10	2			[]
720	Why?	1			[]

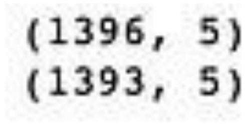
# drop these rows

Print df_training.shape

Df_training = df_training.drop([475, 648, 720])

Df_training = df_training.reset_index(drop=True)

Print df_training.shape



```
(1396, 5)
(1393, 5)
```

Cleaning Prediction Data

# load the data

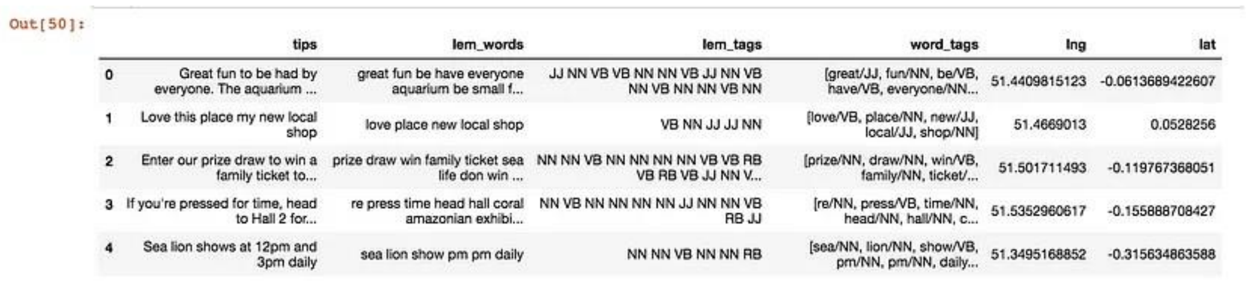
Fs = pd.read_csv('./foursquare/foursquare_csv/londonvenues.csv')

# use cleaning functions on the tips

Word_tag_fs = cleaning_function(fs.tips)

Lemm_fs = separate_word_tag(word_tag_fs)

```
# concat cleaned text with original

Df_fs_predict = pd.concat([fs, lemm_fs], axis=1)

Df_fs_predict[‘word_tags’] = word_tag_fs

# separate the long lat

Lng=[]

Lat=[]

For ll in df_fs_predict[‘ll’]:

   Lnglat = ll.split(‘,’)

   Lng.append(lnglat[0])

   Lat.append(lnglat[1])

Df_fs_predict[‘lng’] =lng

Df_fs_predict[‘lat’] =lat

#  drop the ll column

Df_fs_predict = df_fs_predict.drop([‘ll’], axis=1)

Df_fs_predict.head()
```

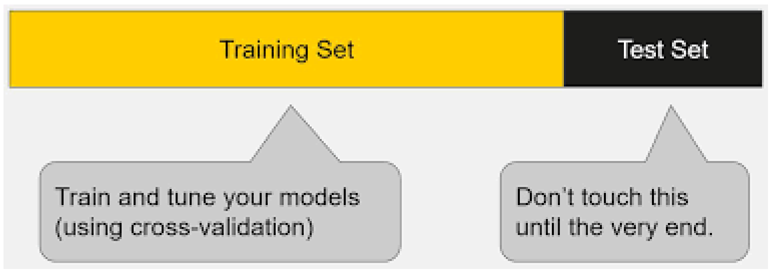


Out[50]:

	tips	lem_words	lem_tags	word_tags	lng	lat
0	Great fun to be had by everyone. The aquarium ...	great fun be have everyone aquarium be small f...	JJ NN VB VB NN NN VB JJ NN VB NN VB NN NN VB NN	[great/JJ, fun/NN, be/VB, have/VB, everyone/NN...	51.4409815123	-0.0613689422607
1	Love this place my new local shop	love place new local shop	VB NN JJ JJ NN	[love/VB, place/NN, new/JJ, local/JJ, shop/NN]	51.4669013	0.0528256
2	Enter our prize draw to win a family ticket to...	prize draw win family ticket sea life don win ...	NN NN VB NN NN NN NN VB VB RB VB RB VB JJ NN V...	[prize/NN, draw/NN, win/VB, family/NN, ticket/...	51.501711493	-0.119767368051
3	If you're pressed for time, head to Hall 2 for...	re press time head hall coral amazonian exhibi...	NN VB NN NN NN NN JJ NN NN VB RB JJ	[re/NN, press/VB, time/NN, head/NN, hall/NN, c...	51.5352960617	-0.155888708427
4	Sea lion shows at 12pm and 3pm daily	sea lion show pm pm daily	NN NN VB NN NN RB	[sea/NN, lion/NN, show/VB, pm/NN, pm/NN, daily...	51.3495168852	-0.315634863588

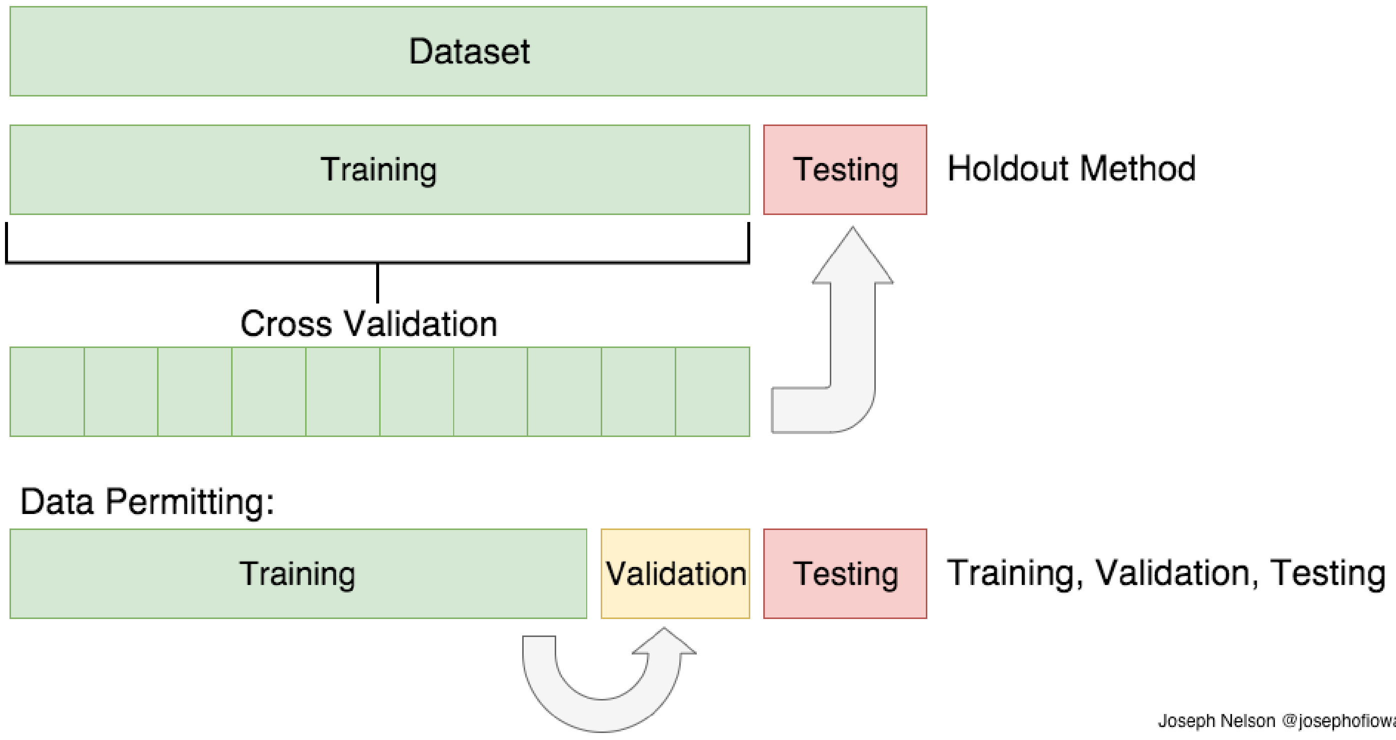
Train Test Split

To measure the accuracy of the model we are creating, the data needs to split into 2 parts. A training set to fit and tune our model and a testing set to create predictions on and evaluate the model at the very end.


Training Set
Test Set
Train and tune your models (using cross-validation)
Don't touch this until the very end.

Validation of the training data

The training set is used to optimise the model using different models and parameters. Typically, it is further separated to train the model and validate on a section which is held out. After we have found the model with the best predictive score through cross validation and tuning parameters we test it against the test set.

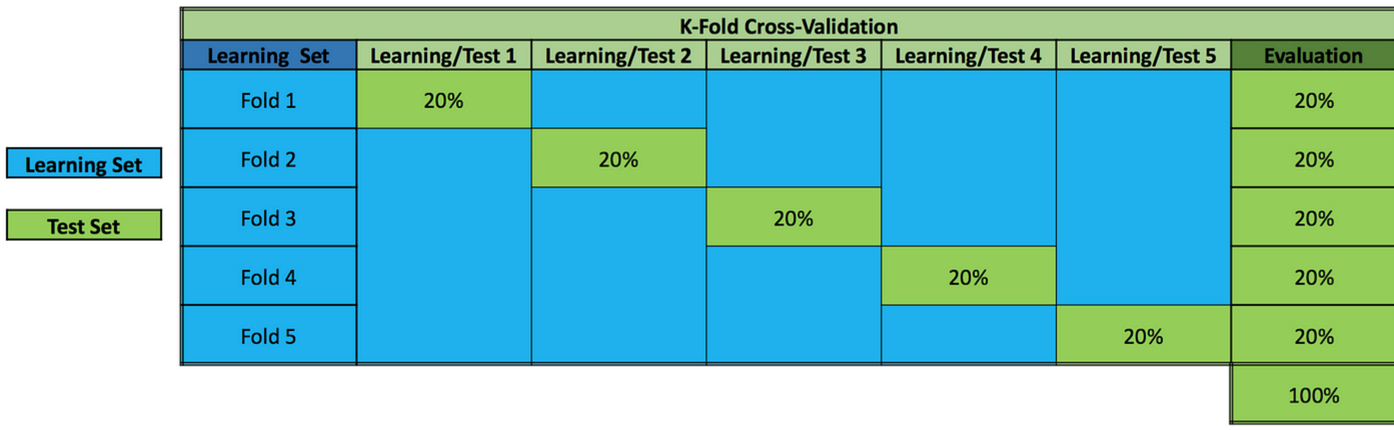


K-Fold Cross Validation

K-fold CV represents the K number of folds/ subsets. Our training set is further split into k subsets where we train on k-1 and test on the subset that is held. This is done for each k fold with a k scores given as a result. We average the model against each of the folds to finalise our model.

By rotating through the subsets of training data it helps the resulting model to generalise (prevent overfitting and underfitting)

.



Learning Set

Test Set

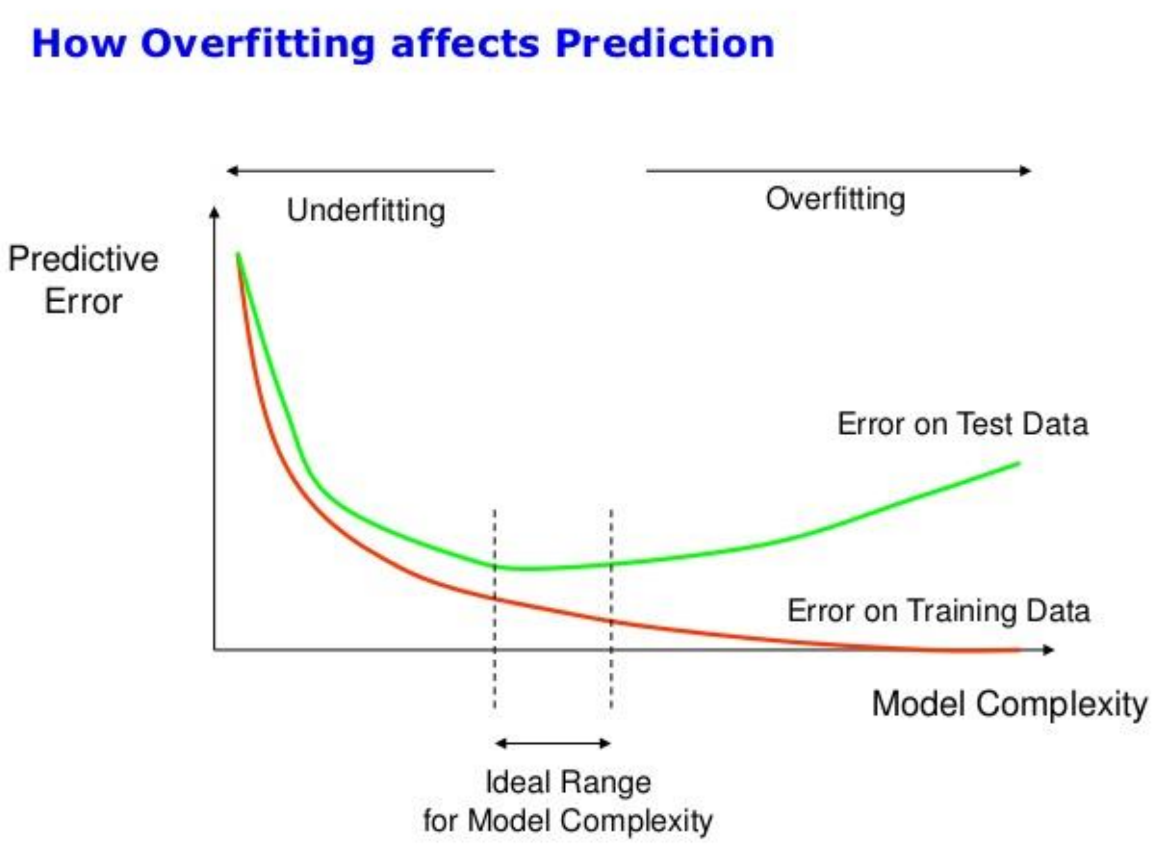
K-Fold Cross-Validation						
Learning Set	Learning/Test 1	Learning/Test 2	Learning/Test 3	Learning/Test 4	Learning/Test 5	Evaluation
Fold 1	20%					20%
Fold 2		20%				20%
Fold 3			20%			20%
Fold 4				20%		20%
Fold 5					20%	20%
						100%

What is Overfitting/Underfitting a Model?

A model that is ungeneralized means you can't make accurate predictions on other data.

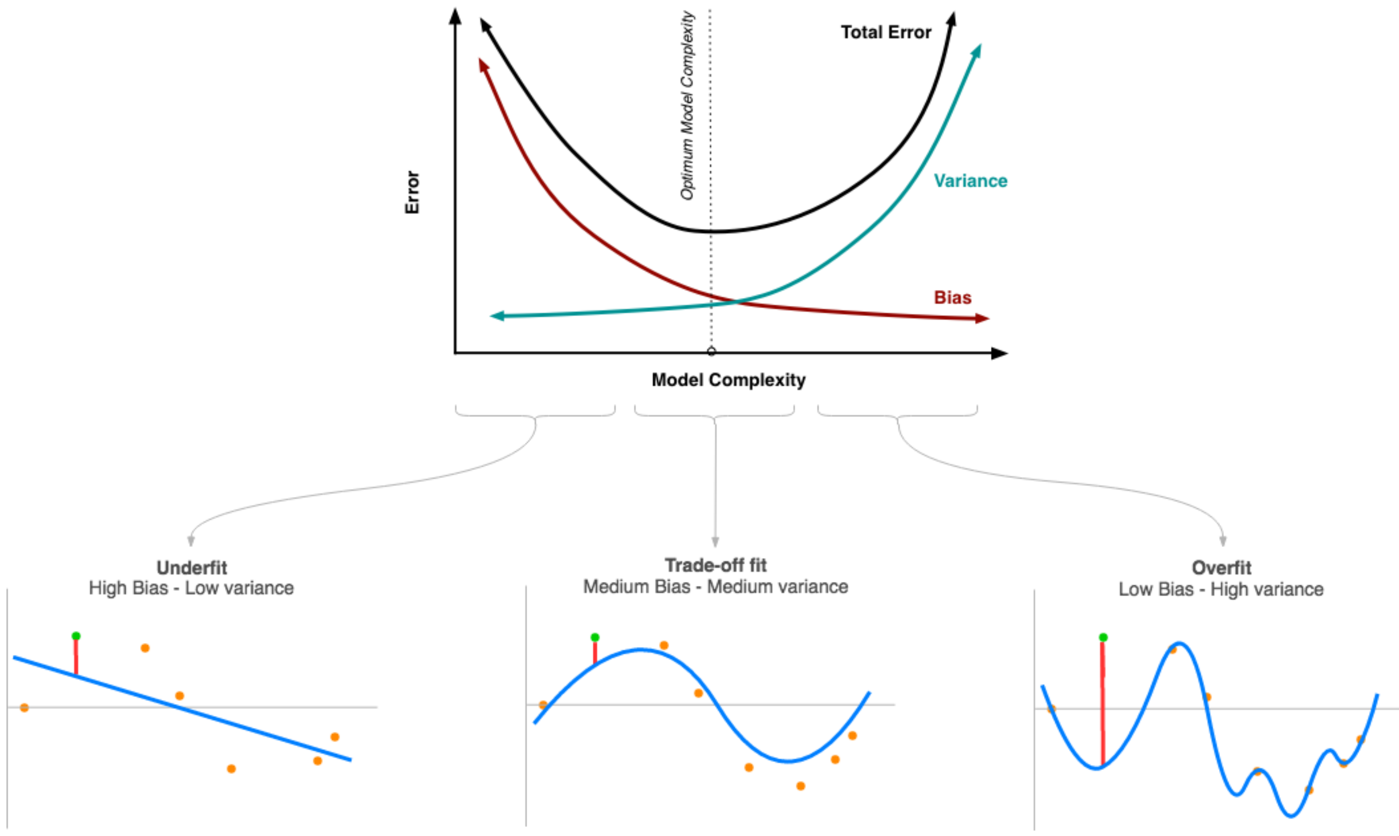
Overfitting is when the model is fit too closely to the training dataset. This is identified when the model is very accurate on the cross validated training data but not very accurate on testing, untrained or new data.

As shown in the image below, overfitting increase the error between the test and training increase. This can be caused by model complexity where there may be too many features/variables compared to the number of observations. Instead of the model learning the actual relationships between variables it learns the noise that is specific to the training set.



Underfitting is when the model does not fit the training data enough and therefore misses the trends in the data. This is identified by the model having a low accuracy score. It can be caused by not enough predictors being used to train the model and therefore is not complex enough. Also if the model chosen is too simple for the data. For example, fitting a linear regression to data that is not linear and so gives poor predictive ability on the training data.

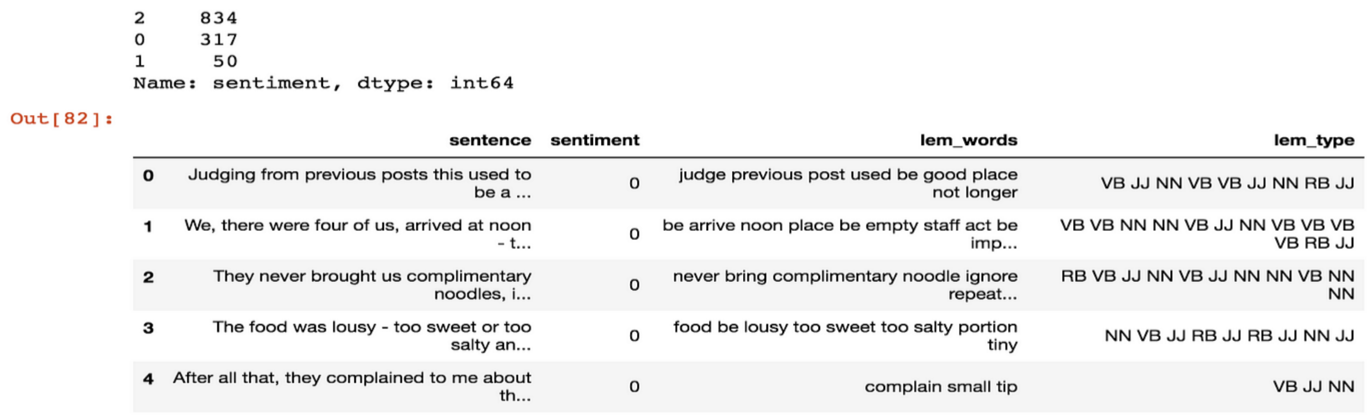
From the image below, you can see the trade off of over and underfitting. We need to create a model which balances both to ensure predictions are accurate.



Back to the Sentiment Classification

Our target variable we are predicting is sentiment.

```
# convert to numeric values
df_training['sentiment'] = df_training.sentiment.map(lambda x:
int(2) if x =='positive' else int(0) if x =='negative' else
int(1) if x == 'neutral' else np.nan)print
df_training['sentiment'].value_counts()
df_training.head()
```



```
2     834
0     317
1      50
Name: sentiment, dtype: int64
```

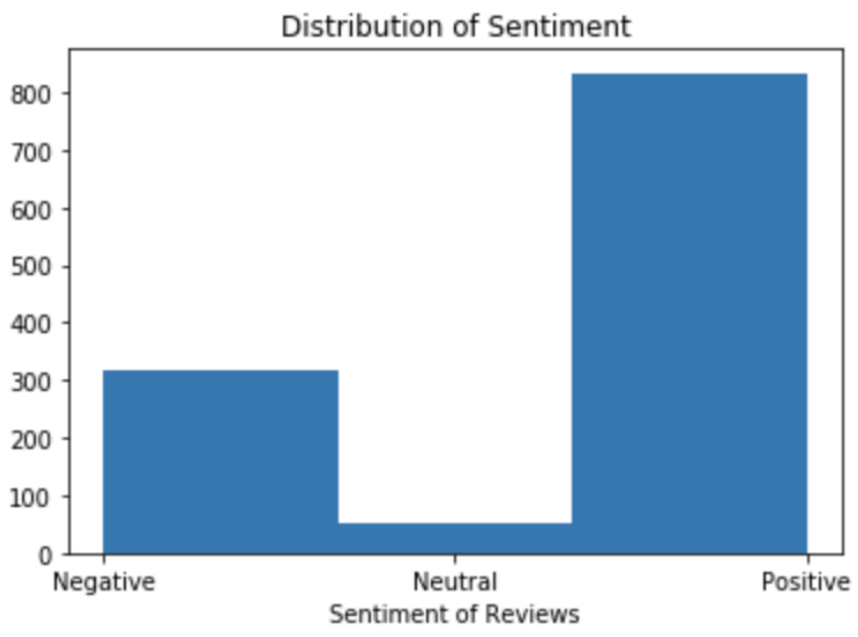
Out[82]:

	sentence	sentiment	lem_words	lem_type
0	Judging from previous posts this used to be a ...	0	judge previous post used be good place not longer	VB JJ NN VB VB JJ NN RB JJ
1	We, there were four of us, arrived at noon - t...	0	be arrive noon place be empty staff act be imp...	VB VB NN NN VB JJ NN VB VB VB VB RB JJ
2	They never brought us complimentary noodles, i...	0	never bring complimentary noodle ignore repeat...	RB VB JJ NN VB JJ NN NN VB NN NN
3	The food was lousy - too sweet or too salty an...	0	food be lousy too sweet too salty portion tiny	NN VB JJ RB JJ RB JJ NN JJ
4	After all that, they complained to me about th...	0	complain small tip	VB JJ NN

Distribution of Sentiment

I first chose to visualise the distribution of my target. The positive class is significantly lager than the other classes. Because of this imbalance, we will need to bootstrap to equalise the baseline accuracy between them. First, we will need to separate the data into training and test sets.

```
import matplotlib.pyplot as plt
plt.hist(df_training.sentiment, bins = 3, align= 'mid')
plt.xticks(range(3), ['Negative','Neutral', 'Positive'])
plt.xlabel('Sentiment of Reviews')
plt.title('Distribution of Sentiment')
plt.show()
```



Train Test Split & Bootstrapping

To evaluate our model we split the data into Training and Testing sets. Here we are using the arguement of test_size = 0.3 which is a ratio of 70/30. The training data will then be used to tune our model through cross validation.

As seen above in the distribution of sentiment the classes are not balanced which can cause problems when measuring the accuracy as each class will have different baseline values. A resampling method with replacement is used, called bootstrapping. The smaller classes are upsampled and the remaining positive class is downsampled to 800 samples each.

```
from sklearn.model_selection import train_test_splittrain,
test = train_test_split(df_training, test_size=0.3,
random_state=1)t_1 =
train[train['sentiment']==1].sample(800,replace=True)
t_2 = train[train['sentiment']==2].sample(800,replace=True)
t_3 = train[train['sentiment']==0].sample(800,replace=True)
```

```
training_bs = pd.concat([t_1, t_2, t_3])print train.shape
print training_bs.shape
print test.shape# sanity check
df_training.shape[0] == (train.shape[0] + test.shape[0])
```

```
        (840, 4)
        (2400, 4)
        (361, 4)

Out[100]: True
```

Baseline Accuracy

The baseline accuracy is the proportion of the majority class. Before bootstrapping ‘2’ which is positive sentiment gives us the baseline at 0.7. After Bootstrapping all the classes the accuracy to predict each of the classes balances so the baseline accuracy is 0.3 for each class.

```
print train['sentiment'].value_counts(normalize=True)
baseline = 0.3
```

```
2    0.697619
0    0.263095
1    0.039286
Name: sentiment, dtype: float64
```

```
print training_bs['sentiment'].value_counts(normalize=True)
baseline = 0.3
```

```
1    0.333333
2    0.333333
0    0.333333
Name: sentiment, dtype: float64
```

Save to csv file

The bootstrap training set and test set is then saved to a csv file, ready for modelling. This will continue in the next post.

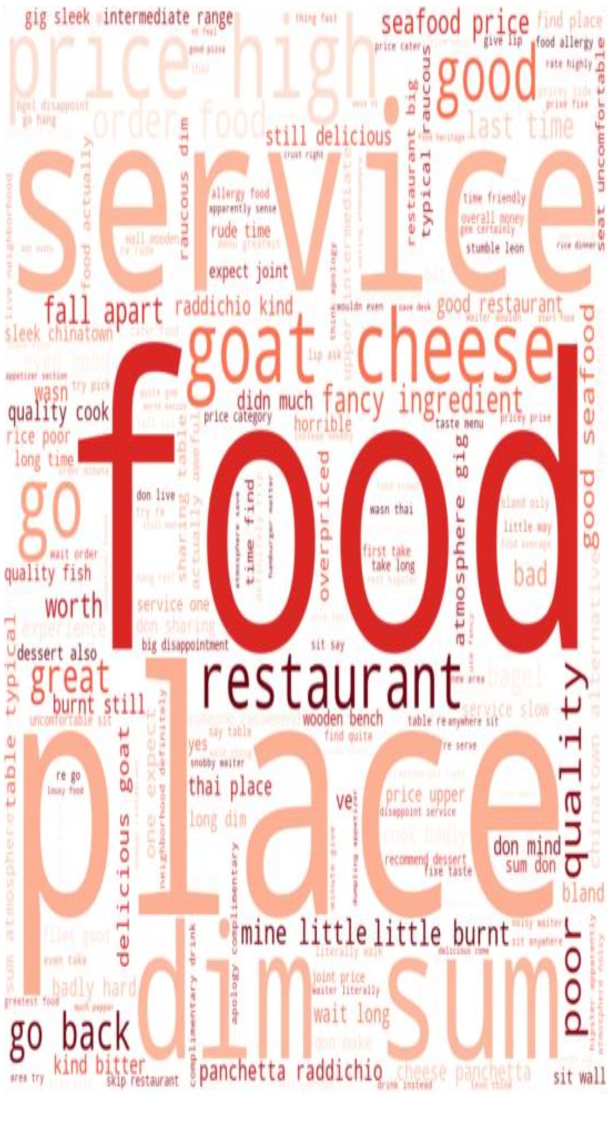
```
# reset index before savingtraining_bs =
training_bs.reset_index(drop=True)
training_bs.to_csv('./train_test_data/training_bs.csv',
header=True, index=False, encoding='UTF8')test =
test.reset_index(drop=True)
test.to_csv('./train_test_data/testing.csv', header=True,
index=False, encoding='UTF8')
```

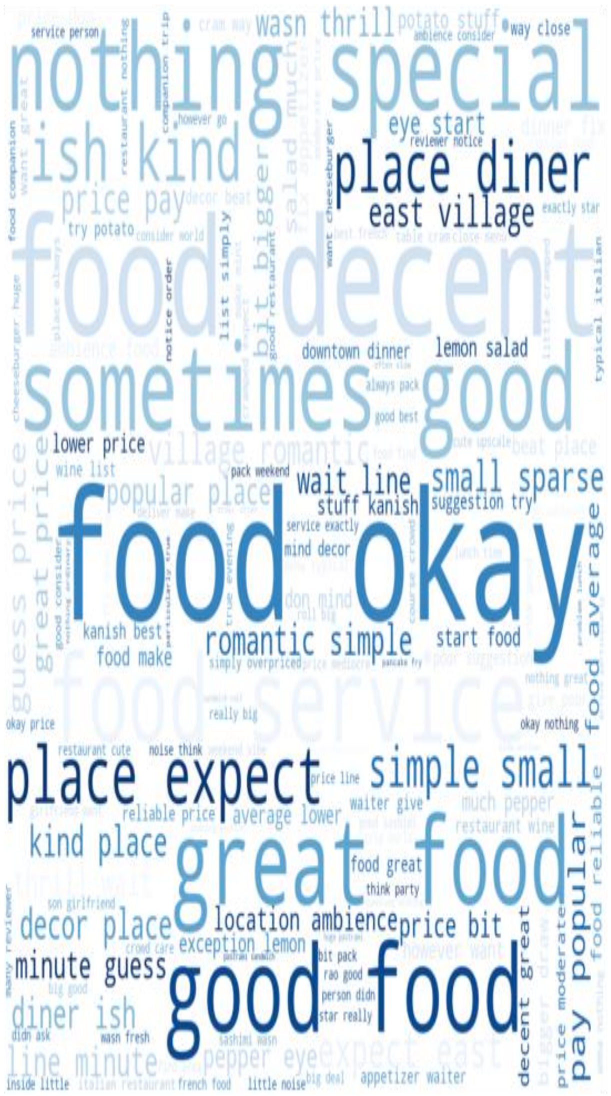
Word Clouds

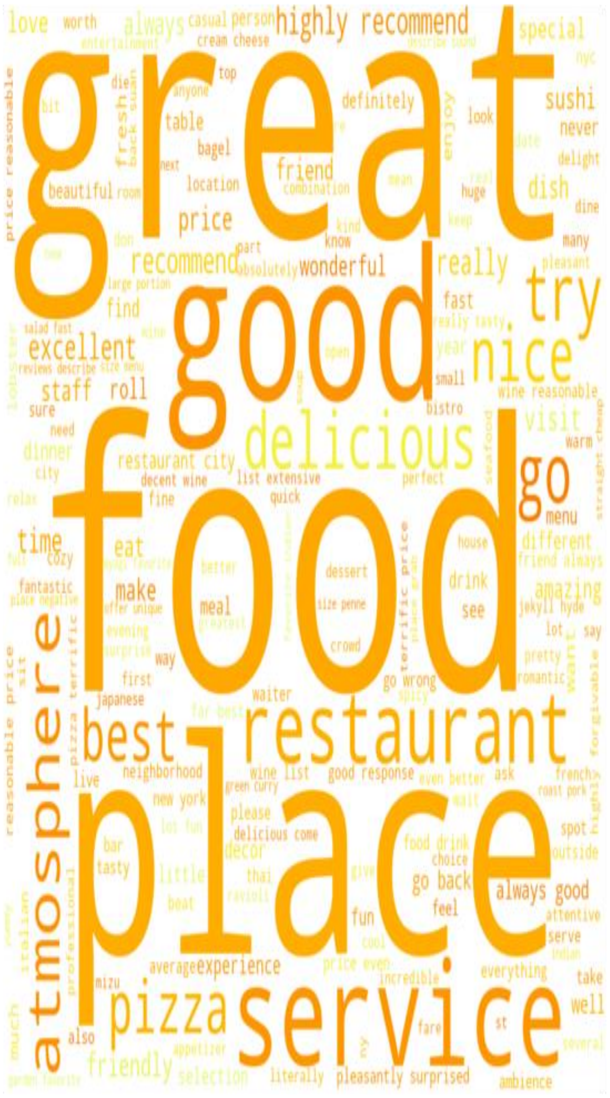
A worldcloud is a collage of randomly arranged words where the size of each word is proportional to its frequency in the corpus. It give us an idea of what words represent in the corpus of each class however, do not clearly indicate accurate information especially when comparing each class.

Just to have an idea of what my final training data looks like, I decided to visualise each class with word clouds. I imported the WordCloud library in python.

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt# Polarity == 0 negative
train_s0 = training_bs[training_bs.sentiment ==0]all_text = ‘
‘.join(word for word in train_s0.lem_words)
wordcloud = WordCloud(colormap=’Reds’, width=1000,
height=1000, mode=’RGBA’,
background_color=’white’).generate(all_text)
plt.figure(figsize=(20,10))
plt.imshow(wordcloud, interpolation=’bilinear’)
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()# Polarity == 1 neutral
train_s1 = training_bs[training_bs.sentiment ==1]all_text = '
'.join(word for word in train_s1.lem_words)
wordcloud = WordCloud(width=1000, height=1000,
colormap='Blues', background_color='white',
mode='RGBA').generate(all_text)
plt.figure( figsize=(20,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()# Polarity == 2 positive
train_s2 = training_bs[training_bs.sentiment ==2]all_text = '
'.join(word for word in train_s2.lem_words)
wordcloud_p2 = WordCloud(width=1000, height=1000,
colormap='Wistia',background_color='white',
mode='RGBA').generate(all_text)
plt.figure(figsize=(20,10))
plt.imshow(wordcloud_p2, interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```




nothing special
food decent
sometimes good
food okay
food service
place expect
great food
good food
place diner
east village
ish kind
price pay
wasn thrill
potato stuff
eye start
small sparse
wait line
popular place
village romantic
romantic simple
simple small
kind place
decor place
minute guess
diner ish
line minute
pepper eye
expect east
location ambience
price bit
start food
stuff kanish
lower price
downtown dinner
lemon salad
food average
pay popular
decent great
price moderate
food reliable
great price
guess price
bit bigger
salad much
list simply
average lower
much pepper
exception lemon
food make
kanish best
food great
way close
service person
cram way
try potato
wine list
okay price
okay nothing
restaurant wine
reliable price
waiter give
appetizer waiter
french food
little noise
inside little
italian restaurant
big deal
rao good
person didn
star really
wasn fresh
didn ask
sashimi wasn
noise think
restaurant cute
really big
suggestion try
service exactly
mind decor
don mind
simply overpriced
typical italian
exactly star
want cheeseburger
cheeseburger huge
food companion
companion trip
restaurant nothing
notice order
true evening
course crowd
good consider
son girlfriend
think party
bit pack
crowd care
many reviewer
however want
pack weekend
lunch time
cute upscale
beat place
nothing great
always pack
good best
reviewer notice
ambience consider
however go
decor beat
consider world
table cram
close menu
want great
place always
ambience food



In the negative cloud some neutral words are big like ‘service’, ‘food’ and ‘place’. Some of the mid sized words are ‘price’ , ‘high’, ‘goat cheese’, ‘dim sum’.

The neutral class larger words ‘food’ ‘okay’, ‘decent’, ‘sometimes’, ‘nothing special’. Giving me an idea of some of the mildly positive and mildly negative reviews.

The positive class has larger words ‘great’ , ‘good’ , ‘food’ and ‘place’. Mid size words that appear are ‘service’, ‘atmosphere’, ‘best restaurant’ and ‘pizza’. Smaller words ‘highly recommended’ , ‘good response’, ‘excellent’, ‘staff’.

‘food’ appears in negative, neutral and positive clouds as one of the biggest sizes. This suggests most people are highlighting their reviews based on the the food. The word ‘place’ is also one of the biggest in both positive and negative which suggests that most non neutral reviews are about the place.