CAPSTONE PROJECT REPORT

Sentiment Analysis of Tweets

The DY Duo

Divya Gladys G divyagladysg@gmail.com,

Yazhini K k.yazhini1971@gmail.com

ABSTRACT

Sentiment analysis is the field of study that analyses people's opinions, sentiments, evaluations, attitudes, and emotions from written language. It is one of the most active research areas in natural language processing. In fact, this research has spread outside of computer science to the management sciences and social sciences due to its importance to business and society as a whole. The growing importance of sentiment analysis coincides with the growth of social media such as reviews, forum discussions, blogs, micro-blogs, Twitter, and social networks.

The aim of this project is to develop a Natural Language Processing (NLP) model that will allow us to do polarity classification. Twitter is a social-networking platform which is widely used and which is a rapidly expanding service. Its magnitude of users is large which result in responses (tweets) which are more convincing and more general. Sentiment Analysis is the process of obtaining opinions from the different polarities (Positive, Negative or Neutral). The nature of the opinions extracted from the tweets can be classified with the help of Sentiment Analysis.

Keywords: Natural Language Processing, Sentiment Analysis, Polarity, Model

INTRODUCTION

Monitoring and analysing opinions from social media provides enormous opportunities for both public and private sectors. For private sectors, it has been observed that the reputation of a certain product or company is highly affected by rumours and negative opinions published and shared among users on social networks. Understanding this observation, companies realize that monitoring and detecting public opinions from micro blogs leads to building better relationships with their customers, better understanding of their customers' needs and better response to changes in the market. For public sectors, recent studies show that there is a strong correlation between activities on social networks and the outcomes of certain political issues. In this project, we focus on doing Sentiment Analysis on Twitter Data (tweets).

We try to classify each tweet as positive, negative or neutral from the given data. The evaluation metric for this project is the mean F1-Score. The given training dataset consists of three columns: tweet_id, sentiment and tweet_text. We need to pre-process this dataset and overcome all the difficulties brought upon during the process so as to apply different interesting models and try to achieve favorable output for the given test dataset by reaching maximum accuracy possible through our model.

DEFINING SENTIMENTS

For the purpose of our project, we define a tweet to have a positive sentiment if the tweet contains positively intended words or emoticons (basically positive notion declared in the sentences of the tweet) or a negative sentiment if it contains words and emoticons which are intended to show negative characteristics. A tweet is considered to be having a neutral sentiment if it is just like a sentence that could be seen in newspaper headlines or in Wikipedia. Below are the examples for the three types of sentiments.

tweet_id	sentiment	tweet_text
263970913270251520	positive	"Good morning Thursday. \"""Life is fragile. We\u2019re not guaranteed a tomorrow so give it everything you\u2019ve got.\""" - Tim Cook [Do it for Jobs!]"
253050608322502657	neutral	The Business Leader\u2019s Award ceremony will be held during the @wef annual meeting in Davos on the 25 January 2013! http://t.co/YI4r8UFM
213342054351257601	negative	Desperation Day (February 13th) the most well known day in all mens life.

Table 1: Sample tweets for each sentiment class from the training Dataset.

DATA SOURCE

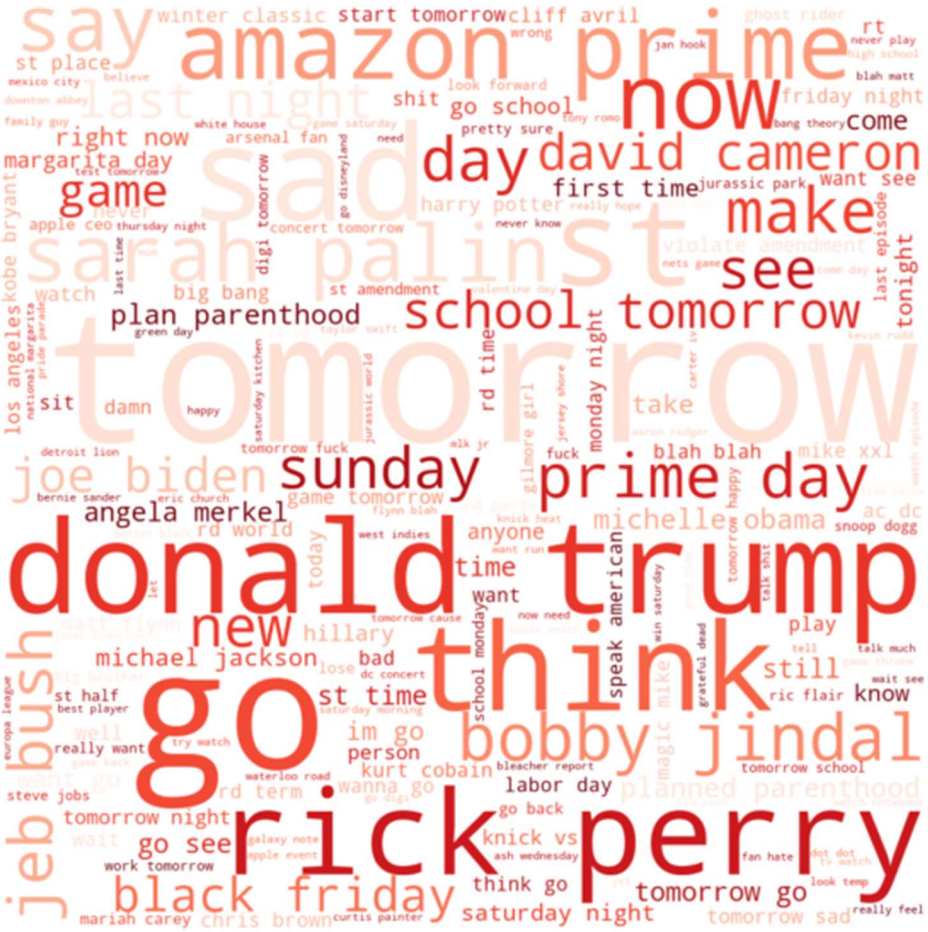
The data set used for this project has been obtained from Kaggle.com.

EXPLORATORY DATA ANALYSIS

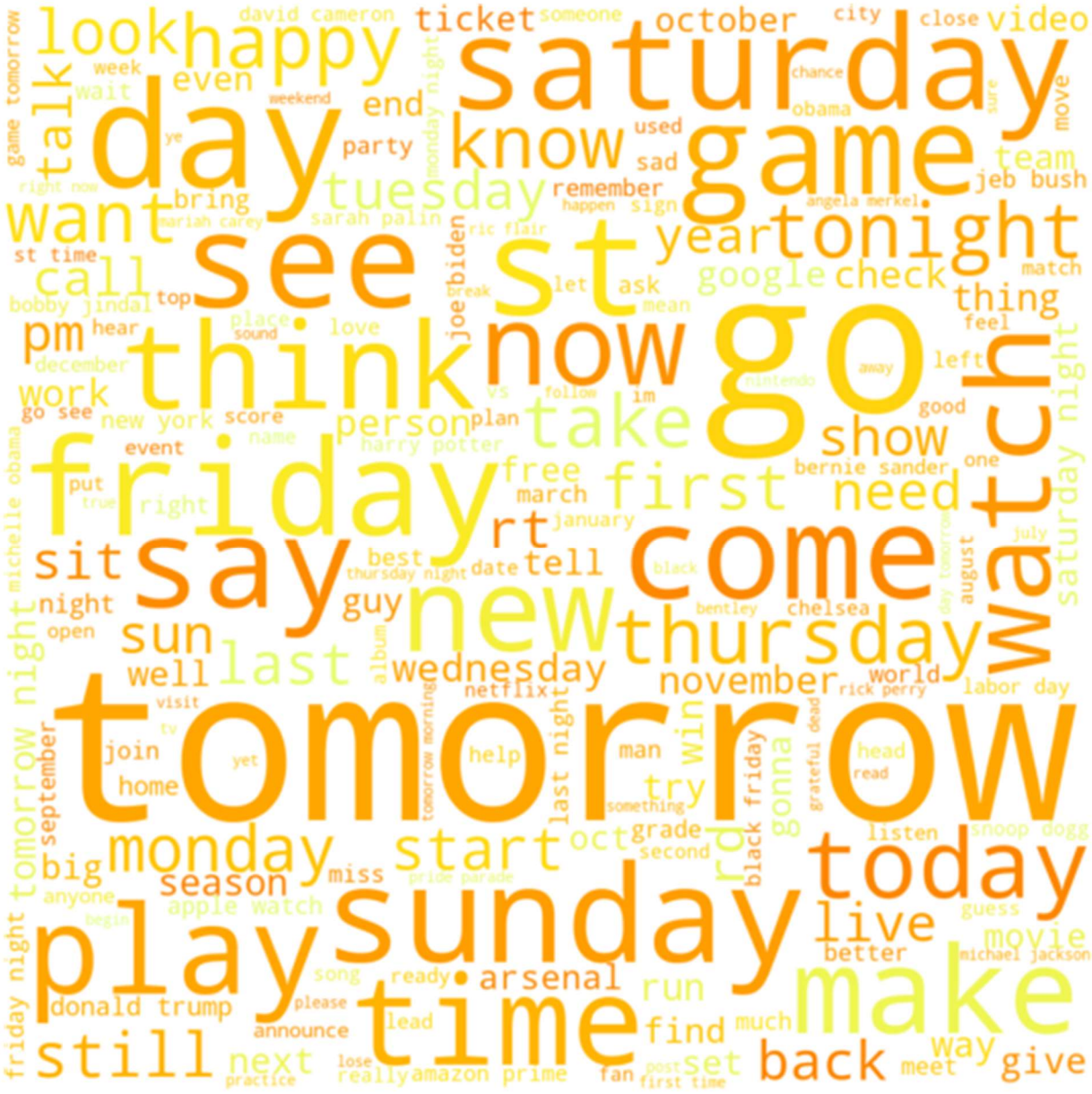
Word Cloud

A **word cloud** is a popular visualization of **words** typically associated with text data used to identify keywords in that text data. They are most commonly used to highlight popular or trending terms based on frequency of use and prominence. A **word cloud** is a beautiful, informative image that communicates much in a single glance.

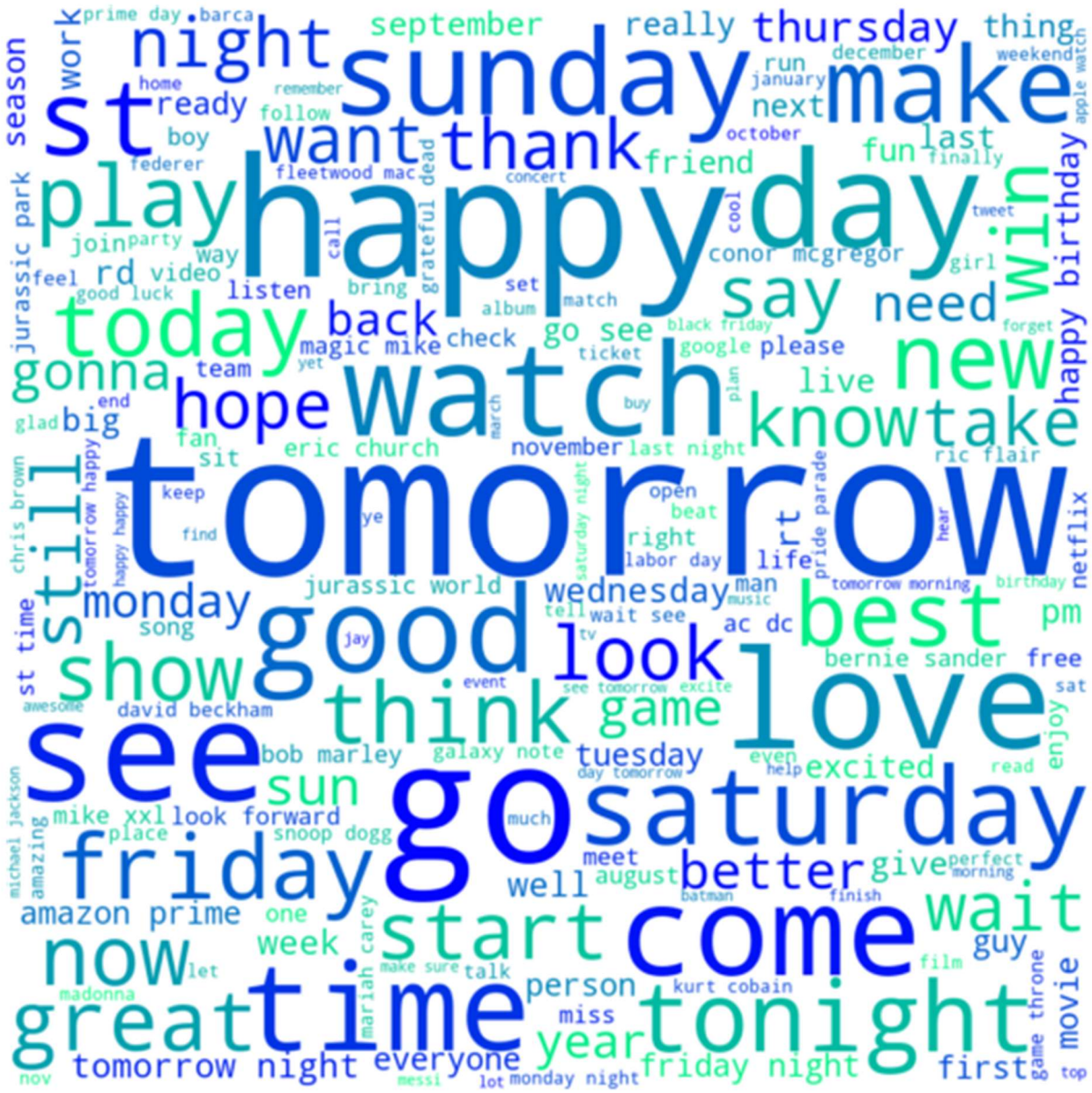
In all of the word cloud images we can see that the words with most frequency contain no emotions in it. We cannot predict the sentiment with those words, for example-Tomorrow, time, amazon prime, etc. So it is better to remove those words which have maximum frequency for better results.



The word cloud above depicts the keywords in the tweets with negative sentiment from the training dataset.



The word cloud above depicts the keywords in the tweets with neutral sentiment from the training dataset.



This word cloud above depicts the keywords in the tweets with positive sentiment from the training dataset.

DATA PRE-PROCESSING

A tweet contains a lot of opinions about the data which are expressed in different ways by different users .The twitter dataset used in this project work is already labelled into three classes such as negative, neutral and positive polarity and thus the sentiment analysis of the data becomes easy to observe the effect of various features. The raw data having polarity is highly susceptible to inconsistency and redundancy.

Pre-processing of tweet include following features:

- Removing all URLs (e.g. www.xyz.com), hash tags (e.g. #topic), targets (@username).
- Correcting the spellings; sequence of repeated characters is removed.
- Replacing all the emoticons with their sentiment.
- Removing all punctuations, symbols, numbers.
- Removing Stop Words.
- Expanding Acronyms.
- Removing Unicode (e.g. \\u002c, etc)
- Lowercasing the tweets.
- Tokenization.
- Normalization.
- Substitution

Removing Unicode characters and # ,@ symbols:

A sample tweet containing Unicode before pre-processing

```
In [19]: tweets=df['tweet_text']
         print(df['tweet_text'][1])
         for i in range(len(tweets)):
              # A : removing html entities like &lt; &gt; &amp;
             A = xml.sax.saxutils.unescape(tweets[i])
             # B : Removing @ tagged words and # tags
             #tweets[i] = " ".join(filter(lambda x:x[0]!='@',tweets[i].split()))
             B = ' '.join(re.sub("(@[A-Za-z0-9]+)|(#[A-Za-z0-9]+)", " ", A).split())
```

```
Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday.
```

The same sample tweet after pre-processing, the Unicode has been removed and the text is converted to lowercase.

```
In [27]: print(tweets[1])

         theo walcott is still shit watch rafa and johnny deal with him on saturday
```

Stop words:

Stop words are the most commonly occurring 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 has provide a library used for this. We have not removed stop words in our project work because they remove words like 'not', etc which indicate the sentiment of the tweet.

Example:

```
In [166]: print(df.tweet_text[2])
          print(df.sentiment[2])

          its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february.
          negative
```

Tokenization:

Tokenization is the process of converting text into tokens before transforming it into vectors. It is also easier to filter out unnecessary tokens. We are tokenising the tweets, which is a sentence, into words.

Normalization:

Words which look different due to casing or written another way but are the same in meaning need to be processed 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 is called normalization.

Casing the Characters:

Converting character to the same case so the same words are recognized as the same. We converted all the letters to lowercase letters to enhance the model.

Removing unnecessary characters:

Standalone punctuations, special characters and numerical tokens are removed as they do not contribute to sentiment which leaves only alphabetic characters.

In this example the punctuation marks, @ tags are all removed.

```
In [41]: print(df['tweet_text'][17800])

@DVATW @helenketting  very destitute with their I phones sat nav an Nike tops ! Some were in Internet cafes in Hungary just had
to check in

In [42]: print(tweets[17800])

very destitute with their i phones sat nav an nike tops some were in internet cafes in hungary just had to check in
```


Lemmatization:

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

```
from gensim.utils import lemmatize
lemm = lemmatize('wow. thank you all for coming out to my younow and playing minecraft with me! so much fun. love you all!')
word = [lemm[i].decode("utf-8-sig") for i in range(len(lemm))]
word
```

```
['thank/NN',
 'come/VB',
 'younow/RB',
 'playing/NN',
 'minecraft/NN',
 'so/RB',
 'much/JJ',
 'fun/NN',
 'love/VB']
```

Defining functions:

We created a cleaning function which will be applied to the whole dataset. It includes decoding, lowercasing, tokenising, removal of special characters, standalone punctuation. A separate function to do lemmatization with POS tagging.

Checking for null and empty values:

There were no null values found in the data before pre-processing. Therefore, the same statistics is maintained even after the cleaning process.

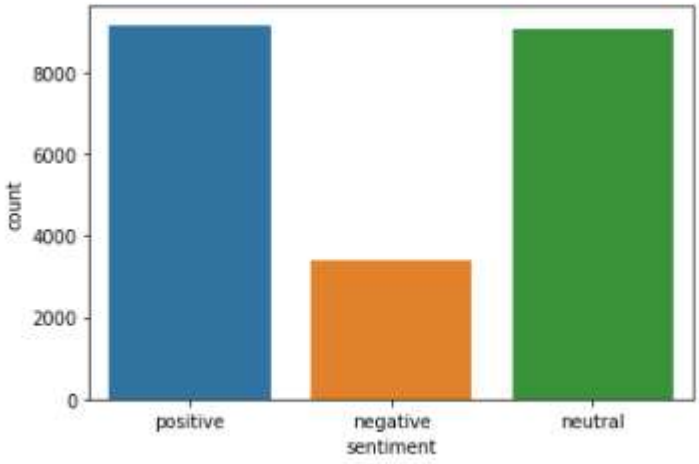
Distribution of Sentiment

We visualised the distribution of the three sentiments. The positive class and the neutral class are significantly larger than the negative class. Because of this imbalance, we had to bootstrap to equalise the baseline accuracy between them.

```
In [20]: print(df.sentiment.value_counts())
         sns.countplot(x="sentiment", data=df_training)
         #Not balanced
```

```
positive    9155
neutral     9075
negative    3400
Name: sentiment, dtype: int64
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x163f7309c88>
```



From the above image it is clearly seen there very few negative sentiment when compared to positive and neutral sentiments.

We have then converted the string type classes of sentiments to numerical values. i.e., we have changed positive → 2, neutral →1 and negative →0 to feed into the classifier.

Train Test Split and Bootstrapping

To evaluate our model, we split the dataset into Training and Testing sets. Here we are using the argument of test size = 0.3 which splits the dataset in ratio 7:3.

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. The smaller classes are up sampled and the remaining positive class is down sampled to 20000 samples each.

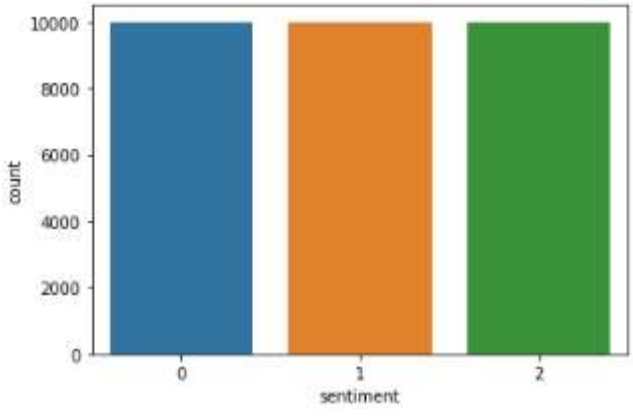
```
In [24]: print("Before bootstrapping")
         print(train['sentiment'].value_counts(normalize=True))
         baseline = 0.3

         print("\n")
         print("After Bootstrapping")
         print(training_bs['sentiment'].value_counts(normalize=True))
         baseline = 0.3
         sns.countplot(x="sentiment", data=training_bs)
```

```
Before bootstrapping
1    0.424411
2    0.420118
0    0.155472
Name: sentiment, dtype: float64


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

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x163f8ef7e10>
```



In the above graph, we can see that all the sentiments are normalized perfectly. Now the data is ready for modelling.

TRAINING THE MODEL

FEATURE EXTRACTION

We experimented with 2 different vectorisers to see which one was best for the data and used a logistic regression model as it is a fast and simple classifier.

TFIDF VECTORIZER

TFIDF or tf–idf, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection of data. It is often used as a weighting factor in searches of information retrieval, text mining, and user modelling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

COUNT VECTORIZER

Bag-of-Words or Bow is a simple model which assigns a unique number to each word in the document so that we can encode any document fixed-length vectors of the length of the vocabulary of the known words. The count or frequency of each word in the encoded corpus (document) fills the value in each position of the vector.

```
df_cvec = pd.DataFrame(X_train_cvec.todense(),columns=cvec.get_feature_names())
print(df_cvec.shape)
df_cvec.head()
```

```
(30000, 15189)
```

	aa	aac	aah	aahh	aale	aalim	aaliyah	aapl	aaron	ab	...	zuckerberg	zuckerman	zulu	zuma	zumba	zumiez	zuoma	zurich	zylona	zz
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
2	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
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 15189 columns

In [108]:
```
print('Baseline:', baseline)
print('Count Vectorizer Score:', cvec_score.mean())
print('Tfidf Vectorizer Score:', tvec_score.mean())

acc_list = []
acc_list.append(cvec_score.mean())
acc_list.append(tvec_score.mean())

# DataFrame Accuracy
acc_df = pd.DataFrame()
acc_df['params']= ['cvec', 'tvec']
acc_df['scores']= acc_list
acc_df
```

```
Baseline: 0.3
Count Vectorizer Score: 0.8453661702902943
Tfidf Vectorizer Score: 0.7809333422844383
```

Out[108]:

	params	scores
0	cvec	0.845366
1	tvec	0.780933

When we compared the accuracy of both the vectorizers, count vectorizer had higher accuracy rate when compared to tf-idf vectorizer.

TUNING HYPER-PARAMETERS

As we had better results for count vectorizer, we tried tuning its parameters for better results. Hyper parameters tune a model from the default conditions. We investigated n-gram range, max features and max df to see which conditions would give us a higher accuracy score.

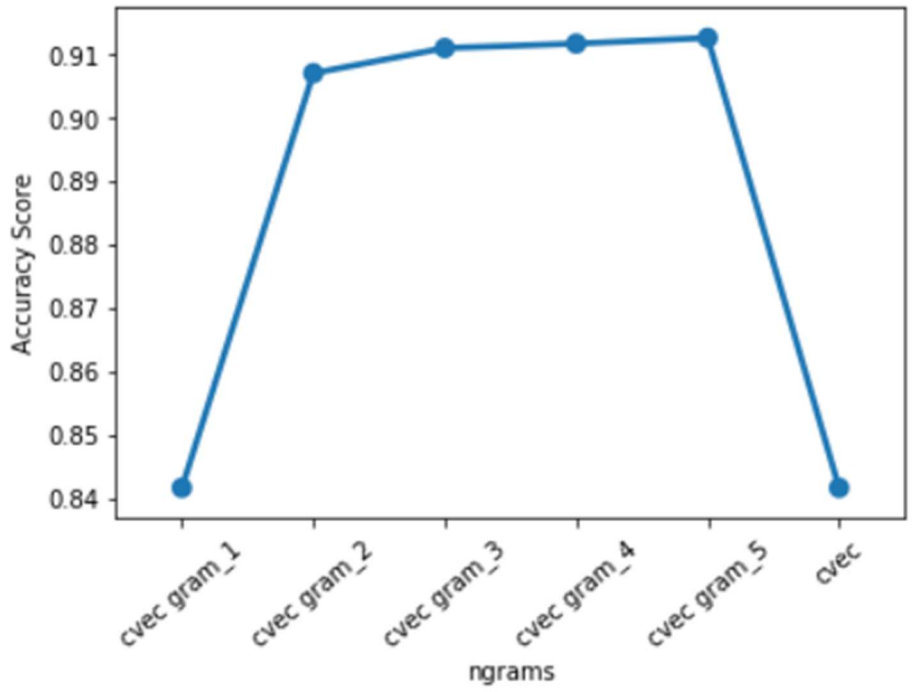
NGRAM:

N-gram range will take a range of n tokens to use as features.

For example,

- (1,1) default: using only a singular token
- (1,2) bigram: using a range of singular and double tokens
- (1,3) trigram: using a range of singular, double and triple tokens

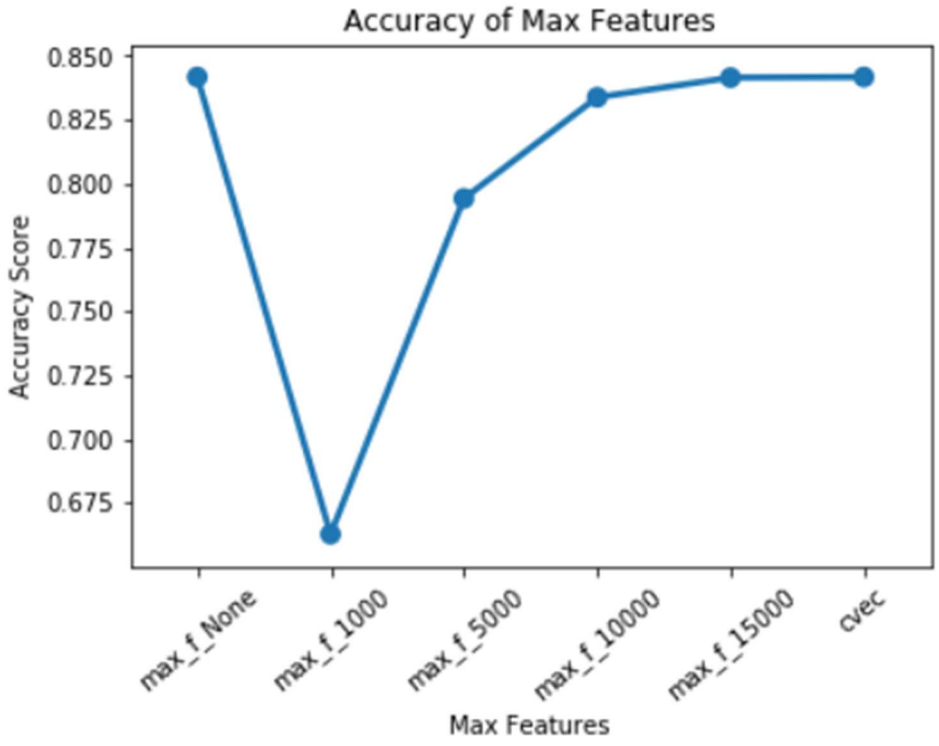
As shown below, it was concluded that the accuracy increases with the range because as it increases it gives more information for the model to predict. Here (1,5) range has the highest score.



Max Features:

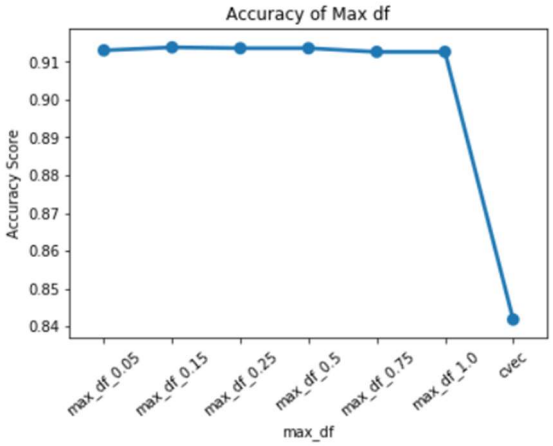
Count vectorizer transforms each token into a predictive feature and with text data this feature size could be in the thousands or even millions. Max Features is a parameter that will limit how many of these we can use as predictors.

Using in conjunction with ngrams, it will also tell us which ones are the most significant with frequency. We investigated a range of max features. Including all the features (default param) the accuracy was at the highest value. This is because as we increase the max features the more words, we give the model to train on thus increasing the accuracy.



MAX DF:

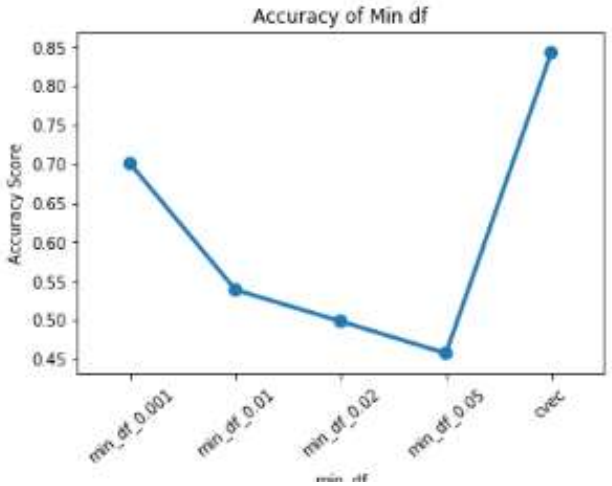
This is the threshold of the Maximum Document frequency of a token. This can help limit the words that may appear too frequently in a corpus.



The accuracy was almost same regardless of the tuning. Therefore, we used the default max df value for this parameter.

MIN DF:

Min_df is used for removing terms that appear too infrequently. In this parameter also , the default value gives the highest accuracy compared to others



CLASSIFIERS:

We separated the data into training and test data in the beginning. Now we apply these classifiers in the training data

We used various types of classifiers to obtain better accuracy. Among the classifiers we used; Simple linear regression provided the best accuracy among all the classifiers.

Below are the accuracy and F-1 score of other classifiers which we used:

1. Logistic regression Cross Validation estimator fit with l1 regularisation:

```
# CV - CrossValidation estimator fit with l1 regularisation
model_l1 = LogisticRegressionCV(Cs=np.logspace(-10,10,21),penalty = 'l1',solver='liblinear',cv=3,multi_class='auto')

lrcv1_score = cross_val_score(model_l1, X_train_cvec, y_train, cv= 3)
print(lrcv1_score.mean())

model_l1.fit(X_train_cvec, y_train)

p1 = model_l1.predict(X_train_cvec)
acc_score_lrcv1 = metrics.accuracy_score(y_train, p1)
f1_score_lrcv1 = metrics.f1_score(y_train, p1, average='macro')

print('Total accuracy classification score: {}'.format(acc_score_lrcv1))
print('Total F1 classification score: {}'.format(f1_score_lrcv1))
```

```
0.8890664111402143
Total accuracy classification score: 0.9982
Total F1 classification score: 0.9982001496253456
```

2. Logistic regression Cross Validation estimator fit with l2 regularisation:

```
model_l2 = LogisticRegressionCV(Cs=np.logspace(-10,10,21), penalty = 'l2',solver='liblinear',cv=3, multi_class='auto')

lrcv2_score = cross_val_score(model_l2, X_train_cvec, y_train, cv=3)
print(lrcv2_score.mean())

model_l2.fit(X_train_cvec, y_train)

p2 = model_l2.predict(X_train_cvec)
acc_score_lrcv2 = metrics.accuracy_score(y_train, p2)
f1_score_lrcv2 = metrics.f1_score(y_train, p2, average='macro')

print('Total accuracy classification score: {}'.format(acc_score_lrcv2))
print('Total F1 classification score: {}'.format(f1_score_lrcv2))
```

```
0.9068664414931794
Total accuracy classification score: 0.9994333333333333
Total F1 classification score: 0.9994333183228331
```

3. Stochastic Gradient Descent with Logistic Regression Classifier

```
## Fit logistic classifier on training data
clf = SGDClassifier(loss="log", penalty="none")
clf.fit(X_train_cvec, y_train)

sgd_score = cross_val_score(clf, X_train_cvec, y_train, cv= 3)
print(sgd_score.mean())

clf.fit(X_train_cvec, y_train)

p1 = clf.predict(X_train_cvec)
acc_score_lrcv1 = metrics.accuracy_score(y_train, p1)
f1_score_lrcv1 = metrics.f1_score(y_train, p1, average='macro')

print('Total accuracy classification score: {}'.format(acc_score_lrcv1))
print('Total F1 classification score: {}'.format(f1_score_lrcv1))
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:166: FutureWarning: max_iter and tol para
meters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not
None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:166: FutureWarning: max_iter and tol para
meters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not
None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
```

```
0.8111657596473504
Total accuracy classification score: 0.9099
Total F1 classification score: 0.9096847437219474
```

4. Naive Bayes Classifier

```
from sklearn.naive_bayes import MultinomialNB
nb_clf = MultinomialNB()
nb_clf.fit(X_train_cvec, y_train)

nb_score = cross_val_score(nb_clf, X_train_cvec, y_train, cv= 3)
print(nb_score.mean())

nb_clf.fit(X_train_cvec, y_train)

p1 = nb_clf.predict(X_train_cvec)
acc_score_nb = metrics.accuracy_score(y_train, p1)
f1_score_nb = metrics.f1_score(y_train, p1, average='macro')

print('Total accuracy classification score: {}'.format(acc_score_nb))
print('Total F1 classification score: {}'.format(f1_score_nb))
```

```
0.8877001377402287
Total accuracy classification score: 0.9959
Total F1 classification score: 0.9958979547223364
```

5. Stochastic Gradient Descent with Hinge and L2 regularisation.

```
from sklearn.linear_model import SGDClassifier
svm_clf = SGDClassifier(loss="hinge", penalty='l2')
svm_clf.fit(X_train_cvec, y_train)

svm_score = cross_val_score(svm_clf, X_train_cvec, y_train, cv= 3)
print(svm_score.mean())

svm_clf.fit(X_train_cvec, y_train)

y_pred = svm_clf.predict(X_train_cvec)
acc_score_svm = metrics.accuracy_score(y_train, y_pred)
f1_score_svm = metrics.f1_score(y_train, p1, average='macro')

print('Total accuracy classification score: {}'.format(acc_score_svm))
print('Total F1 classification score: {}'.format(f1_score_svm))
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:166: FutureWarning: max_iter and tol para
meters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not
None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
```

```
0.8244670697823633
Total accuracy classification score: 0.9203333333333333
Total F1 classification score: 0.9096847437219474
```

Below is the comparison of training cross validation accuracies of the various classifiers:

```
print('Logistic regression:', lr_score.mean())
print('Logistic regression CrossValidation estimator fit with l1 regularisation:', lrcv1_score.mean())
print('Logistic regression CrossValidation estimator fit with l2 regularisation:', lrcv2_score.mean())
print('Stochastic Gradient Descent with Logistic Regression Classifier:',sgd_score.mean())
print('Naive Bayes Classifier:',nb_score.mean())
print('Stochastic Gradient Descent with Hinge and L2 regularisation:',svm_score.mean())

acc_list = []
acc_list.append(lr_score.mean())
acc_list.append(lrcv1_score.mean())
acc_list.append(lrcv2_score.mean())
acc_list.append(sgd_score.mean())
acc_list.append(nb_score.mean())
acc_list.append(svm_score.mean())

# DataFrame Accuracy
acc_df = pd.DataFrame()
acc_df['params']= ['Logistic regression','Logistic regression CV with l1 ',
                   'Logistic regression CV with l2 ',
                  'Stochastic Gradient Descent + Logistic Classifier','Naive Bayes Classifier',
                   'Stochastic Gradient Descent + Hinge & L2 ']
acc_df['scores']= acc_list
acc_df
```

```
Logistic regression: 0.9089333248461818
Logistic regression CrossValidation estimator fit with l1 regularisation: 0.9040002080591982
Logistic regression CrossValidation estimator fit with l2 regularisation: 0.9068664414931794
Stochastic Gradient Descent with Logistic Regression Classifier: 0.8964333445942074
Naive Bayes Classifier: 0.8877001377402287
Stochastic Gradient Descent with Hinge and L2 regularisation: 0.9083667314941852
```

	params	scores
0	Logistic regression	0.908933
1	Logistic regression CV with l1	0.904000
2	Logistic regression CV with l2	0.906866
3	Stochastic Gradient Descent + Logistic Classifier	0.896433
4	Naive Bayes Classifier	0.887700
5	Stochastic Gradient Descent + Hinge & L2	0.908367

TESTING THE MODEL:

Now we are using the model we created on the test data.

MODEL 1:

As already found, we used the count vectorizer with customized tuning of hyper parameters and simple linear regression model on the test data.

```
pred = lr.predict(X_test_cvec)
acc_score = metrics.accuracy_score(y_test, pred)
f1_score = metrics.f1_score(y_test, pred, average='macro')

print('Total accuracy classification score: {}'.format(acc_score))
print('Total F1 classification score: {}'.format(f1_score))
```

```
Total accuracy classification score: 0.6264447526583449
Total F1 classification score: 0.5892596148988551
```

Thus, we obtained 0.62 accuracy on the validation dataset.

MODEL 2:

Fasttext:

fastText is a library for learning of word embeddings and text classification created by Facebook's AI Research (FAIR) lab. The model allows creating an unsupervised learning or supervised learning algorithm for obtaining vector representations for words. Facebook makes available pre-trained models for 294 languages. fastText uses a neural network for word embedding. We have used this model to predict sentiments.

This second model provides us with more accuracy than the previous model with an accuracy of 0.77 on the validation dataset

```
model = fasttext.train_supervised(input='fasttext_train.txt')
model_acc_training_set = model.test('fasttext_train.txt')
model_acc_validation_set = model.test('fasttext_valid.txt')

# DISPLAY ACCURACY OF TRAINED MODEL
text_line = "accuracy:" + str(model_acc_training_set[1])  + ", validation:" + str(model_acc_validation_set[1]) + '\n'
print(text_line)
```

```
accuracy:0.8935833333333333, validation:0.7739251040221914
```

The submission of our predictions of the test-samples provided in the same source (Kaggle.com) where we took our training dataset yielded 66% as our F1-score for our MODEL 1 and 67% as our F1-score for MODEL 2

CONCLUSION

We have used many machine learning algorithms with the help of various packages widely available today. This project also created in us the thirst to venture in other fields such as Neural Networks where LSTM (Long short-term memory) models are considered to further enhance sentiment analysis.

Application of sentiment analysis on tweets is a challenging issue. Our results provide some evidence that with comprehensive pre-processing effort and machine learning algorithms, sentiment analyses can be performed on tweets. Furthermore, adding a third neutral value to negative positive poles can enrich the scale for precision purposes. The results of this study provide support to the argument that sentiment analysis can be used to complement survey-based attitude research.

References:

1. Sentiment analysis on twitter
   https://www.rcciit.org/students_projects/projects/it/2018/GR33.pdf
2. Using machine learning for sentiment analysis: a deep dive
   https://algorithmia.com/blog/using-machine-learning-for-sentiment-analysis-a-deep-dive
3. A General Approach to Preprocessing Text Data
   https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html
4. Steps for effective text data cleaning (with case study using Python)
   https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/?#
5. Sentiment analysis of reviews: Text Pre-processing
   https://medium.com/@annabiancajones/sentiment-analysis-of-reviews-text-pre-processing-6359343784fb
6. Twitter Sentiment Analysis using fastText
   https://towardsdatascience.com/twitter-sentiment-analysis-using-fasttext-9ccd04465597
7. NLP Sentiment Analysis Handbook
   https://towardsdatascience.com/nlp-sentiment-analysis-for-beginners-e7897f976897
8. Another Twitter sentiment analysis with Python — Part 3 (Zipf's Law, data visualisation)
   https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-3-zipfs-law-data-visualisation-fc9eadda71e7