Identify the sentiment by Babatunde Oginni [05/08/2019]

**Executive Summary**The report presented here is based on the analysis of a learning data sets which is available on the Analytics Vidhya website, it was meant to identify sentiments in tweets. This report highlights the various methodologies used in the analysis of the tweets. As at the time of writing this report, the best submission obtained by the author of this report had an F1-score (evaluation metric used) of 0.8975 which is less than 0.02 difference from the F1-score of the top place position(0.9164) on the public leader board; this submission was placed in the top 15th percentile on the public leader board. The best result presented in this report was obtained by combining two approaches, both of which are based on the Multinomial Naïve Bayes algorithm. The tokenization used max\_df of 0.5 and ‘english’ stop words; two tokenization processes were used – splitting each tweet into single words and lemmatization of the each non-stop words in the tweets. This report also highlights the results of various techniques used which did not result in better performance on the public leader board but could be useful in the analysis of some other data sets.

**Introduction**The data analyzed in this report was obtained from Analytics Vidhya, the link is: <https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/>

An excerpt of the problem statement reads: “*given the tweets from customers about various tech firms who manufacture and sell mobiles, computers, laptops, etc, the task is to identify if the tweets have a negative sentiment towards such companies or products*.” The evaluation metric used to determine the performance of classification model was weighted F1-score.  
F1-score is one of the classification metrics that can be used to determine the performance of a model; it is the harmonic mean of precision and recall. It can be interpreted as a weighted average of the precision and recall [1]. The best value of F1-score is 1 and the worst is 0. The formula for the F1-score is:  
 F1-score = 2 \* (precision \* recall) / (precision + recall)

Weighted F1-score uses the class count to normalize the evaluation of the harmonic average of recall and precision. A summary of the description of the metrics that are used in classification problems in terms of confusion matrix and associated metrics - precision, recall, f1-score and accuracy can be found in the data school documentation [2]. As at the time of putting this report together (05/06/2019), there were 253 teams/individuals that have registered for this learning competition. The scores on the public leader board are: 1st – 0.9164, 25th – 0.8988, 50th – 0.8937, 75th – 0.8879 while at 100th – 0.8834; difference of less than 0.04 in F1-scores among the top 100 positions. The best submission based on the analysis presented in this report was 0.8975 which placed 33rd on the public leader board. This report is divided into various analysis sections, each section placed emphasis on various methodologies explored. The introductory section highlight the initial analysis which explored the data by looking at the top 50 words that are likely to result in the classification of a particular class of sentiments; the first section showed model optimization where different models were studied and the parameters tuned to enhance performance; second section used model ensemble where an attempt was made to see the effect of combining different models of various predicted probabilities to see if it could result in better performance, section three explored the analysis of some engineered features and the performance on the test data, fourth section showed the use of stemming and lemmatization on the words of the tweet while the last section tested a number of possibilities including the use of Vader sentiments, word2vec and TextBlob for analysis. The codes associated with the analyzed sections and the data set are on the author’s Github account [3].  
Identification of sentiments has a lot of business applications such as employee engagement survey, customers’ appreciation (or lack thereof) of new and existing products from an organization, monitoring the public perspective of a business brand, or products, monitoring of customers’ reaction to changes in business product or services etc.

**Initial Analysis**

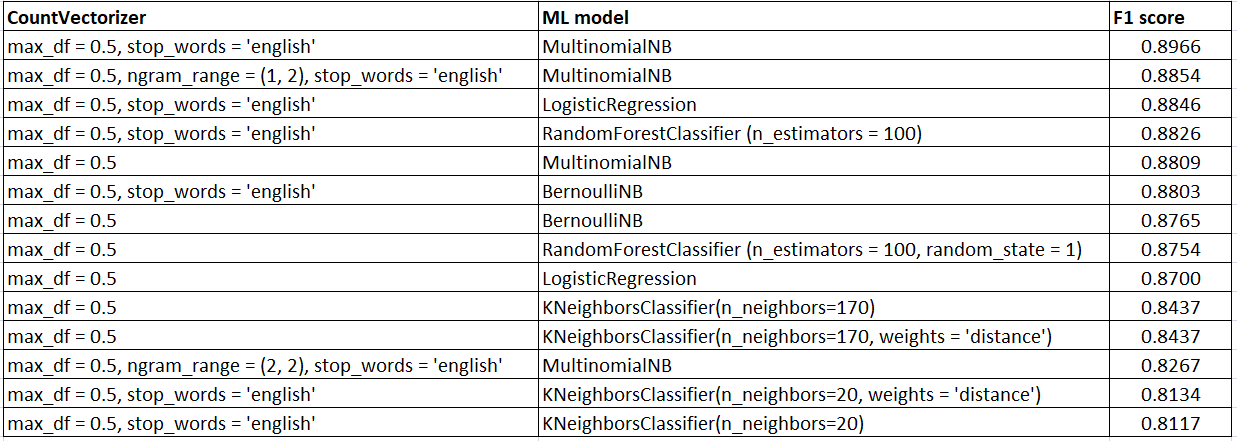
The problem gave train and test data sets; the train data set has 7920 rows and 3 columns while the test data set has 1953 row and 2 columns. The columns are id, label and tweet; the label column was absent in the test data. Analysis was done on the training data set alone; CountVectorizer was used to split the content of the tweet column into words and transformed into a document-term matrix. The dimension of the document-term matrix was (7920, 23090) where the first entry corresponds to the number of rows which is also the number of documents in the training set while the second entry represents the number of unique words in all the training data set. Though the matrix has a very large dimension, only 137638 elements are non-zeros corresponding to about 0.075% non-zero entries; analysis therefore proceeded only on the sparse matrix.

A Multinomial naïve bayes algorithm can be used to determine the frequency of occurrence of the 23090 vocabularies in each class or label; this is done by fitting the algorithm where the sparse matrix represents the independent variables and the label column represents the dependent variable. Each entry is increased by 1 in order to eliminate possible zero entry and then dividing by the class count so as to account for imbalance in the class ratio. Of the 7920 labels in the training data set, 5894 are in class 0 while 2026 are in class 1. The ratios of the normalized frequencies of one class with respect to the other are a good estimate of the probabilities of occurrence vocabularies in a particular class. These ratios can now be arranged in ascending and descending order to be able to have an overview of the vocabularies in each class. This analysis showed clearly that class 1 represents the negative sentiment while class 0 represents the positive sentiment; it also shows that the customers are largely discussing Apple products. This information was not absolutely given in the problem statement but this quick analysis was able to throw light on it. Some of the words that have high probabilities of being a good predictor for class 1 are vulgar in nature and would not be repeated here but a look at the Jupyter notebook of the analysis will show that. In order to have better context of the words that may contribute to each class, a repeat of the class probability estimation mentioned above can be determined by setting the CountVectorizer parameter n-grams = (2, 2), this will allow the analysis of phrases of two subsequent words. The two most predictive phrases for class 1 are “hate apple” and “you suck” while “iphone sougofollow” and “follow me” are the most predictive for class 0. Having a broad knowledge of the content of the data always help in the different kinds of analysis that are possible. The details of the code used in this section are available in Part-0.ipynb of the author’s Github account [3].

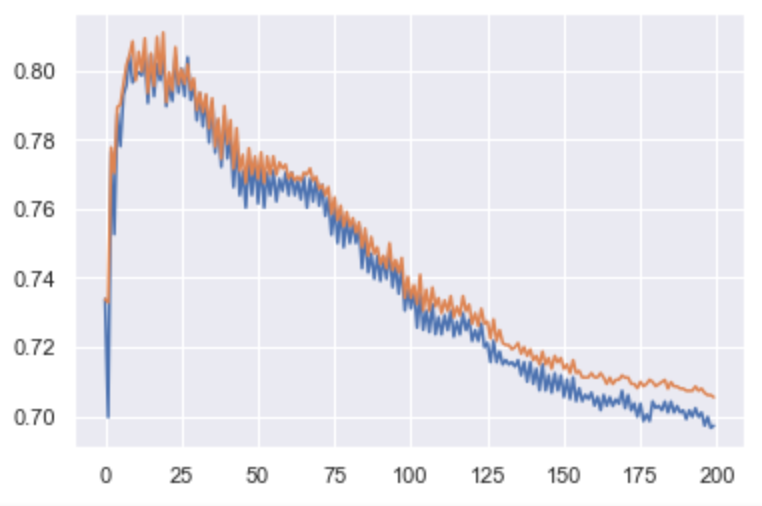
**Optimizing the different models**A grid search on a number of parameters was done so as to determine what the best set of parameters would be based on the analysis of the training data set. The parameters of the CountVectorizer were given a number of possibilities. The stop words which are words that appears too frequently in many documents such as many prepositions, but may not necessarily improve the predictive ability, could be set to None (default) or ‘english’ (where such words are neglected); the tokens could be given option of being lower case or remained they appeared in the documents; token size (ngram\_range) could be individuals words or up to two consecutive words as a phrase; min\_df (words that rarely appear in the entire training set) could be set to 1 or 2 while the max\_df (words that appear too frequently in the document) was could also be varied from say 0.3 up to 1.0. The algorithm for model training used in the grid search was the Multinomial naïve bayes and its alpha (smoothening) parameters could take values of say 1, 0.1 or 0.01. The grid search was performed on a cross-validation of 3 segments (or portions) with the metrics evaluation score set to F1 score. The grid search studied performed 648 cases based on the different parameters ranges stated above. The best parameters were with CountVectorizer with lowercase enabled, max\_df set to 0.5, min\_df set to 1, ngram\_range of (1, 1) and default stop words while the Naïve bayes algorithm had an alpha parameter of unity.

The best set of parameters obtained from the grid search was used to predict the labels of the test data set and the resulting F1-score was 0.8809. It was however discovered that while manually varying the parameters and using the parameters to predict new labels from the test data sets, one could obtain a better F1-score which lead to higher rank on the public leader board. This is not unexpected because the F1-scores are very close and the difference in scores may not be statistically significant. However, in practice whenever the best result from a grid search do not lead to the best predicted result when applied to a test data set, it could possibly mean that the distribution of the training data is different from the distribution of the test data. One could avoid this by mixing the training with test data sets and then shuffling them together before splitting.

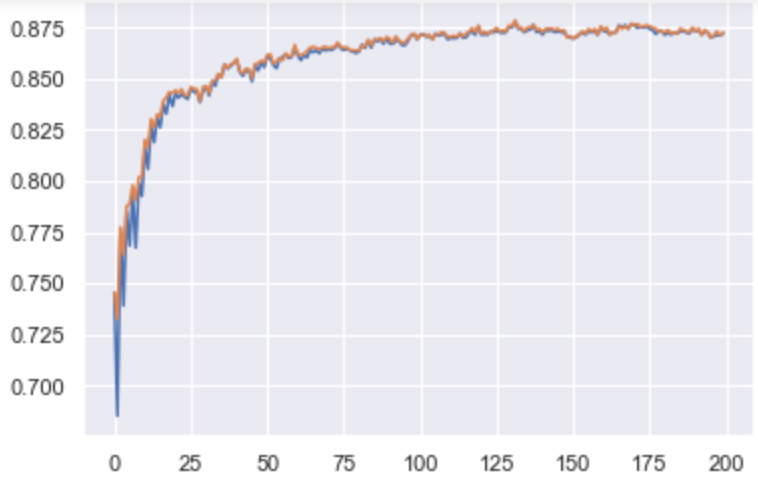
In order to make submission of results faster and check the results of various model and tokenization, a function was defined (named “model\_test”, please see the Jupyter notebook - Part\_1\_Grid\_MNB.ipynb) that has the CountVectorizer (for tokenization of tweets) and a machine learning classifier algorithm as inputs. It fits all the training data sets then predicts labels using the test data set. The output of the function is a csv file which is in the same form as what the submission result should be. The table below shows the various models that were used and the corresponding F1 score obtained for different submissions. This has been arranged in descending order of the F1 scores. The hyper parameters used were the default values otherwise it is stated in the Table.

****

For the KNeighborsClassifier, the n\_neighbors (a measure of how large the nearest neighbors are in order to determine the class label) was determined by computing the metrics for n\_neighbors of various values. In order to do this, the training data set was divided into two portions where the first part trains and fit the model while the F1 score is determined for the other part. The CountVectorizer with stop\_words set to ‘english’ was used to generate the plots below:

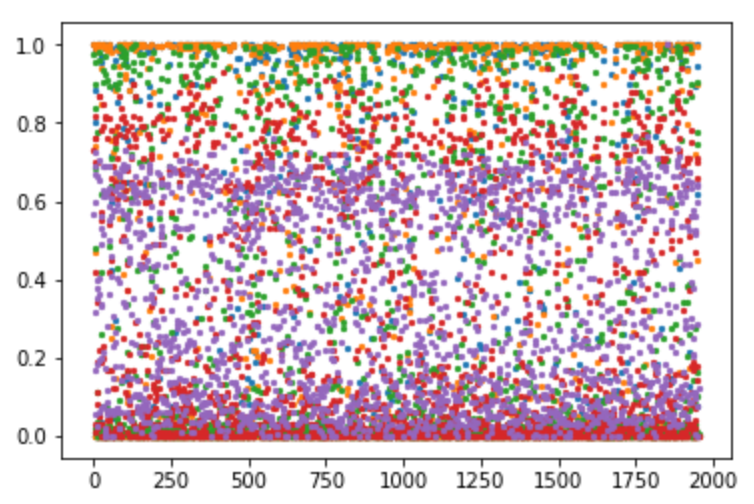


while the plots below were generated with the stop\_words of the CountVectorizer set to None.



The n\_neighbors with the highest F1–score is then used on the test data set. The details of the python scripts that was used to generate this section can be found in Part\_1\_Grid\_MNB.ipynb in the author’s github page [3].

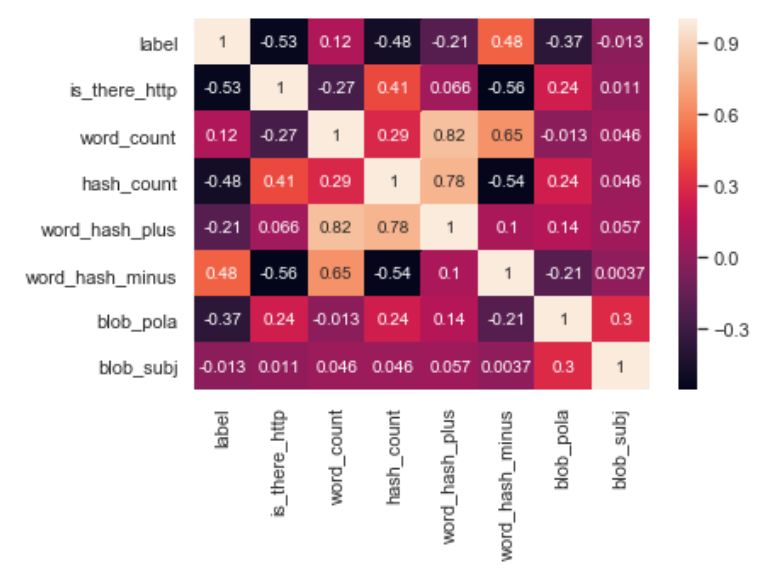
**Model Ensemble**In the previous section, various machine learning algorithms were used to train the tokenized tweets. In addition, the probability of predicting the class labels of each tweet from the test data can be determined for each of the model configuration. The figure below shows the probabilities that the tweets from the test data belong to class 1 for the various algorithms considered; this clearly showed that the different algorithm predict different probabilities; the different color depict a different algorithm.



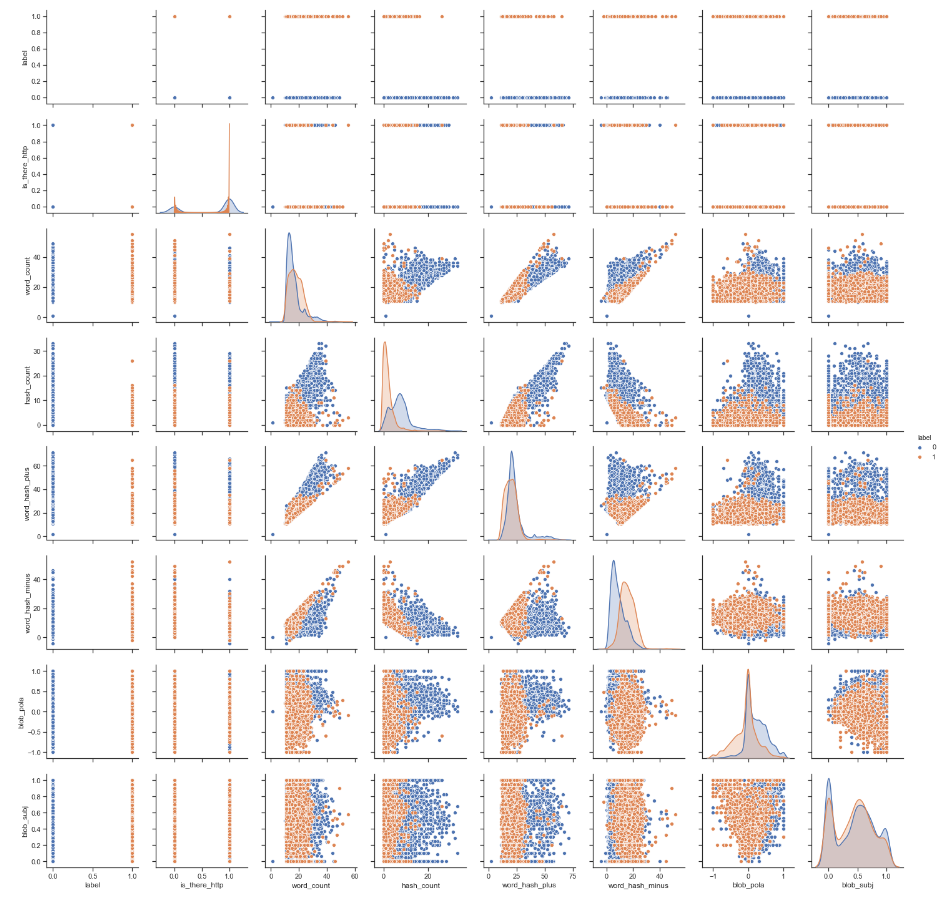
The difference in the predictions and the differences in the theory behind the different algorithms is a possible indication that one can check the linear combinations of the probabilities from the algorithms to see if the submitted result could give a better F1-score. The algorithms considered are Multinomial naïve bayes, Bernoulli naïve bayes, Logistic Regression, Random Forest and KNeighbors. A lot of the weighted factors were varied which resulted in different values for the weighted average probabilities. None of these values gave a better F1-score than the best result from the previous section. In addition, the threshold value of 0.5 was varied to check if the submitted F1-score could be improved but it did not. Finally, a majority vote base for the predicted class from each of the algorithm was considered, in order to check if the weighted aggregate of the class predictions would lead to a better F1-score but it did not. This section involved evaluating a lot of different linear combinations of the models but did not produce a better result than previously observed. The details of python scripts used to study this section are in Part\_2\_Model\_Ensemble.ipynb which is in the author’s github page [3].

**Feature Selections Analysis**In this study, possible feature were identified and investigated. The number of words/tokens in each tweet was determined; it is possible that tweets of negative sentiments may be much longer because the writer may show all sorts of emotions. In similar manner, the number of hash-tags in each tweet could also be a good predictor of sentiment because of writer’s emotion. The sum and difference of the number of words and the number of hash-tags can also be used as engineered features. In addition, checking out the first ten tweets in the training data set, it was noticed that tweets with “http” were all classified as class 0 while those that did not were classified as class 1. There is no clear reason for this observation but it is worth checking out, it can easily be dropped if it is more of noise rather signals. Lastly, Textblob (Python library for processing textual data [4]) methods were applied on each tweet; this can be used to predict the tweet’s polarity and subjectivity. Each of these engineered features can all be represented in numeric forms and one can easily apply Machine Learning classifiers to investigate if the correct classes are predicted.

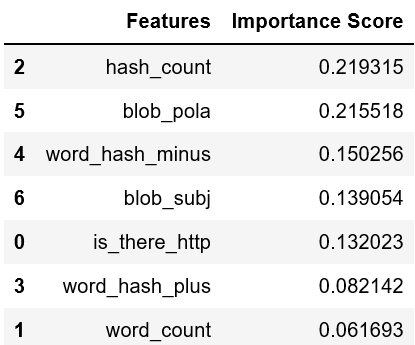
In order to check if any of the features may be redundant, a heat map of the feature’s correlation was determined. The figure below shows the correlation, the word\_count and the hash\_count are fairly correlated with the word\_hash\_plus. Since, the number of features under consideration is not many, and the correlations among the features are less than 0.9, one may not consider dropping any of the features. In other words, all the features can be used as inputs in any of the machine learning algorithms.



A pairplot (plots which show all the features plotted against one another) can also be checked to see any trend, or clear segmentation. It appears that there is quite a lot of overlap in the features and the class labels.



RandomForestClassifier algorithm can be used to determine the importance of each of the features to the prediction of the class lables. The table below shows the features arranged in descending order of the importance score as predicted by the algorithm.



In order to determine the number of features to use in the machine learning algorithm. The training data set was divided into two portions, so that a quick evaluation using train/test split can be performed. The features included was increased in the order decreasing importance as suggested by their RandomForestClassifier. The results are shown in the Table below, the five features that gave the highest F1-score are **'hash\_count', 'blob\_pola', 'word\_hash\_minus', 'blob\_subj', 'is\_there\_http'** which stands for number of hash-tags, polarity of sentiment from TextBlob, number of words minus number of hash-tags, subjectivity of sentiment from TextBlob, ‘http’ present in tweet respectively.

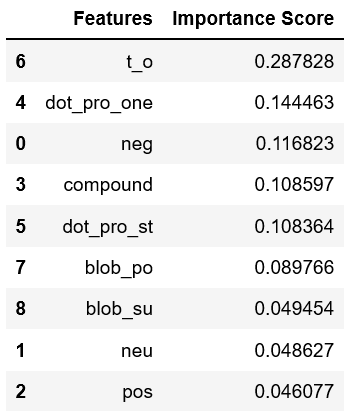
|  |  |
| --- | --- |
| **Features** | **F1-score** |
| 'hash\_count' | 0.8164 |
| 'hash\_count', 'blob\_pola' | 0.8342 |
| 'hash\_count', 'blob\_pola', 'word\_hash\_minus' | 0.8293 |
| 'hash\_count', 'blob\_pola', 'word\_hash\_minus', 'blob\_subj' | 0.8468 |
| **'hash\_count', 'blob\_pola', 'word\_hash\_minus', 'blob\_subj', 'is\_there\_http'** | **0.8659** |
| 'hash\_count', 'blob\_pola', 'word\_hash\_minus', 'blob\_subj', 'is\_there\_http', 'word\_hash\_plus' | 0.8609 |
| 'hash\_count', 'blob\_pola', 'word\_hash\_minus', 'blob\_subj', 'is\_there\_http', 'word\_hash\_plus', 'word\_count' | 0.8607 |

Grid Search was then used to tune a number of machine learning algorithms which were applied on the five selected features described above. These were now applied to the given test data set to determine performance; the best parameters and the corresponding F1-scores are shown below. The details of the python scripts used in the study of this section is in Part\_3\_FeatureEngineering.ipynb of the author’s github page [3].

|  |  |
| --- | --- |
| **Algorithm** | **F1-score** |
| RandomForestClassifier(n\_estimators=500) | 0.8558 |
| GradientBoostingClassifier(n\_estimators=100) | 0.8516 |
| AdaBoostClassifier(algorithm='SAMME.R', n\_estimators=100) | 0.8512 |
| SVC(C=10, kernel='rbf') | 0.8485 |
| KNeighborsClassifier(n\_neighbors=35) | 0.8407 |
| LinearDiscriminantAnalysis(solver='svd') | 0.8290 |
| BernoulliNB(alpha = 0.5) | 0.8156 |
| NuSVC(kernel='rbf', nu=0.5) | 0.8061 |

**Stemming and Lemmatization**  
Up till now, the tokens (single words or group of words) used for studies were on the actual tweets. The probabilities of two different words with the same root form belonging to a class label could potentially be different. It is therefore imperative to check the possibility of setting all the words in the tweets to their root form before tokenization. Two possible ways of reducing words to their roots are through stemming and lemmatization. Thus, each word of the tweets, apart from the ‘english’ stops words, were converted to their root forms using the stemming and lemmatization. All the various techniques that have been discussed in previous sections were used to determine their performance. In addition, the possibility of obtaining a better performance through a linear combination of good models was also explored. The F1-scores associated with the different trials are stated as comments in the Jupyter notebook. The best result obtained by the author was a combination of two models – CountVectorizer on single words of each tweet with max\_df set to 0.5 and ‘english’ as stop words; and CountVectorizer on the root words of each word of tweets obtained from lemmatization with max\_df set to 0.5 and ‘english’ as stop words; the two models made use of Multinomial naïve bayes algorithms. The F1-score obtained when applied on the test data was 0.8975. The details of the python scripts used for the study of this section is in Part\_4\_Stem\_Lemma\_04302019.ipynb in the author’s github page [3].

**Engineered Features using VADER, word2vec and TextBlob**  
This section explores the sentiment based on some Natural Language Processing analysis tools. VADER is the acronym for Valence Aware Dictionary and sEntiment Reasoner and is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media [5, 6]. This tool could be implemented on the tweets to give four possible parameters – negative, neutral, positive and compound; this should convey the inherent sentiments in the tweet. The thought is that these parameters might be good predictor of the class label. The tool was applied to each tweet which subsequently generated four features which can be used as inputs into a Machine Learning classifier. Another tool studied in this section was word2vec; it takes a text corpus as input and produces the word vectors as output [7]. For each tweet, the vector representation is expected to be an average of the vector representation of all the words in the tweet. The thought is that it is possible for tweets of certain class label to produce similar vector representation. In the analysis, the average vector representation for all the training tweets belonging to each class label was determined. The cosine of the angles between theses mean vectors and each vector representation of the individual tweets are supposed to give an idea of their similarities. This idea is also explored in the analysis leading to a number of features that were studied. Lastly, the subjectivity and polarity of each tweet based on the Textblob are further explored in this section. Textblob has earlier been explored in previous section. In all, thirteen features were derived from the tools which were subsequently reduced to nine after correlation analysis of the features. The importance of each of the nine features to the class label prediction was determined using Random Forest Classifier. The table below lists the nine derived features in descending order of importance to the class label prediction. The feature representation are as follows: t\_o is a measure of the angle between the vector representation of each tweet and the average vector representation of tweets in class 0; neg, neu, pos and compound are the negative, neutral, positive and compound scores respectively for the VADER polarity scores on each tweet; blob\_po and blob\_su are polarities and subjectivity scores respectively from the Textblob sentiment polarity; dot\_pro\_one is the dot product of the vector representation of each tweet and the average vector representation of tweets in class label one while dot\_pro\_st is the length of the vector representation of each tweet. The optimum number of features to include in a classifier was now determined by using different subsets of the features in a train/test split analysis with a Random Forest Classifier with 500 desion trees. The analysis suggest that the top seven features would give the best performance which was now used for basis for further analysis. A grid seach of different hyper-parameters was now used on various algorithmms in order to determine the optimum parameters to be used. Finally, each of the best grid search parameters for the different algorithms was now used to evaluate the test data set; this gave different results which are stated as comments in the python scripts. None gave a better result than was previously reported. Although, the analysis technique explored in this section did not result in good predictions but the method could possibly be applied in other data set. The details of the python scripts used in this analysis is in Part\_5\_Vader.ipynb of the author’s github page [3].



**Conclusion**This report is based on the analysis done by the author on the data set provided on the Analytics Vidhya website for the determination of Sentiments in tweets. A number of analytics techniques have been used to study the sentiments of the tweets. Different tokeninzation of the tweets were explored with the inclusion or exclusion of stop words and other parameters; various features were engineered and studied; and various machine learning algorithms were explored. In addition, determining the root words of the tweets using stemming and lemmatization before Count Vectorization was explored. Other natural language processing tools for sentiment predictions like VADER, TextBlob and word2vec were studied as well. Finally, different model ensembles were explored. As at the time of writing this report, the submission that gave the best performance by the author generated an F1-score of 0.8975 on the public test data, which was a difference of less than 0.02 score when compared to the best result on the public leader board.

**Future Work**Further analysis can be done on this data set. An exploration of the sentiments using deep learning techniques could be considered. In addition, a comprehensive exploration of the misclassified tweets during exploration of the training data set could also be done.

**References**

[1.] <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html>  
 sklearn documentation accessed on April 13, 2019

[2.] <https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/>  
 accessed on May 6, 2019

[3.] <https://github.com/tunde-gm3/Sentiment>  
 accessed on May 6, 2019

[4.] <https://textblob.readthedocs.io/en/dev/>  
 accessed on April 22, 2019

[5.] <https://www.kaggle.com/nltkdata/vader-lexicon>  
 accessed on May 8, 2019

[6.] <https://pythonprogramming.net/sentiment-analysis-python-textblob-vader/>  
 accessed on May 8, 2019

[7.] <https://code.google.com/archive/p/word2vec/>  
 accessed on May 8, 2019