Identify the sentiment

by Babatunde Oginni

**Executive Summary**The report presented here is based on the analysis of a learning data sets published on the Analytics Vidhya website which was meant to identify sentiments in tweets. It 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 0.8966 and was placed in the top 15th percentile on the public leader board. The best result presented in this report was obtained by tokenizing the tweets into single words using sklearn’s Count Vectorizer with ‘english’ stop words and max\_df of 0.5, in conjunction with a Multinomial Naïve Bayes algorithm. 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. It also states other techniques that will be explored in future analysis of the data set.

**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. Appendix 1 shows a summary of the description of the metrics that are frequently used in classification problems in terms of confusion matrix and associated metrics - precision, recall, f1-score and accuracy. As at the time of putting this report together (04/08/2019), there were 215 teams/individuals that have registered for this learning competition. The scores on the public leader board are: 1st – 0.9164, 25th – 0.8985, 50th – 0.8925, 75th – 0.8863 while at 100th – 0.8813; difference of less than 0.04 in F1-score between the top 100 positions. The best submission based on the analysis presented in this report was 0.8966 which placed 33rd on the public leader board. This report is divided into four analysis sections – Initial analysis which explored the data by looking at the top 50 words that are likely to result in the classification of given sentiments; it model optimization where different models were studied and the parameters tuned to enhance performance; model ensemble where an attempt was made to see the effect of combine different models of various probabilities to see if it could result in better performance and lastly the analysis of some engineered feature. The codes associated with the analyzed sections and the data set are on Github.  
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**

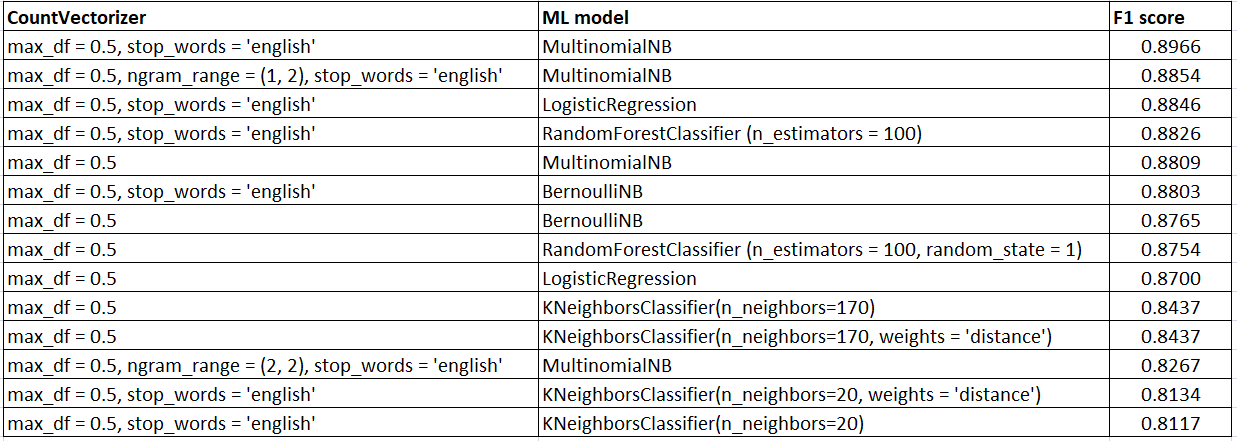
The problem gave train and test data sets; the train data set has 7920 rows and 3 columns while the test data set has 1953 row and 2 columns. The columns are id, label and tweet; the label column was absent in the test data. Analysis was done on the training data set alone; CountVectorizer method was 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 o 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 to 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.

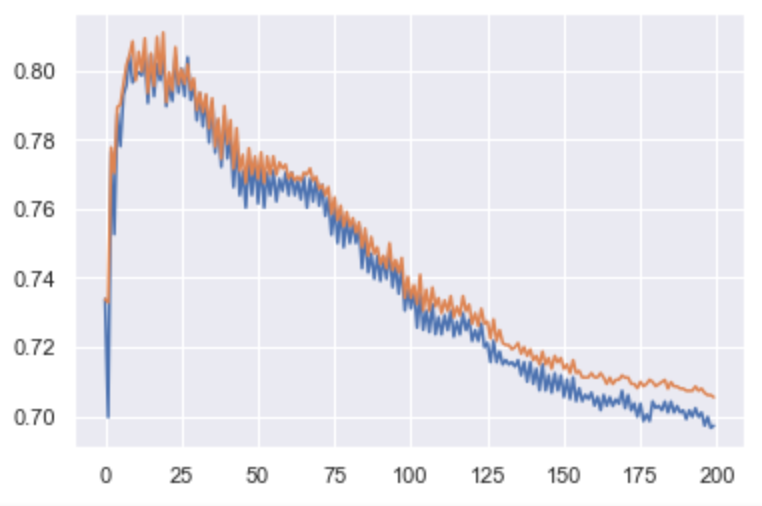
**Optimizing the different models**A grid search on a number of parameters was done so as to determine what the best set of parameters would be based on the analysis of the training data set. The parameters of the CountVectorizer were given different possibilities. The stop words which are words that appears too frequently in many documents, this typically do not make the fitting better were set to None (default) and ‘english’; the tokens were 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) was set to 1 or 2 while the max\_df (words that appear too frequently in the document) was set to 0.3, 0.5 or 1.0. The algorithm for model training used in the grid search was the Multinomial naïve bayes and its alpha parameters could take values of 1, 0.1 or 0.01. The grid search was performed on a cross-validation of 3 with the metrics evaluation score set to F1 score. The grid search performed 648 cases based on the 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.

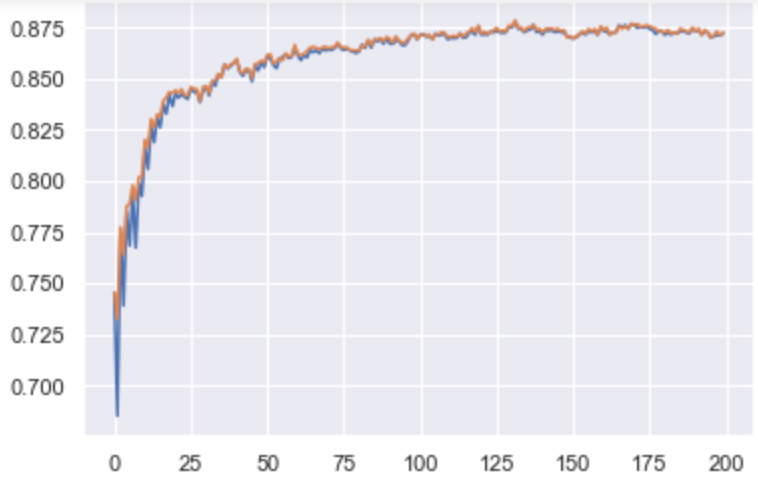
The best set of parameters obtained from the grid search was used to predict the labels of the test data set and the resulting F1-score was 0.8809. It was however discovered that while manually varying the parameters and using the parameters to predict new labels from the test data sets, one could obtain 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 different 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 means 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) that has the CountVectorizer and a machine learning classifier algorithm as inputs. It fits all the training data sets then predict 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. This has been arranged in descending order of the F1 scores. The parameters used were the default values otherwise it is stated in the Table.

****

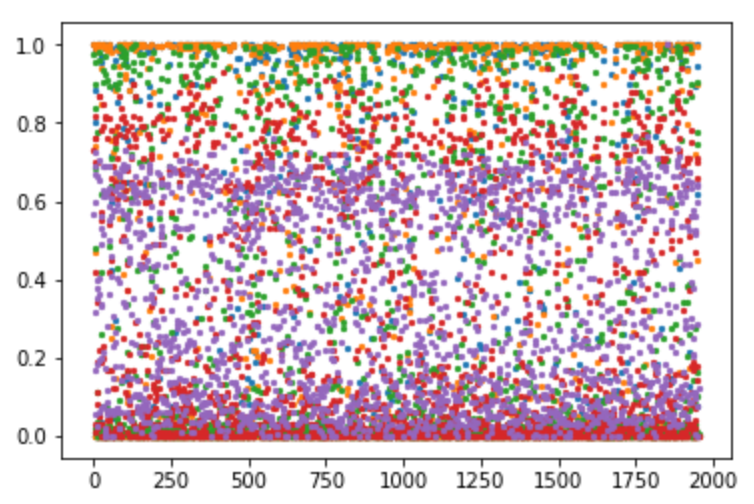
For the KNeighborsClassifier, the n\_neighbors (a measure of how large the nearest neighbors are in order to determine the label) was determined computing the metrics for n\_neighbors of various values. In order to do this, the training data set was diving into two portions where the first part trains 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.

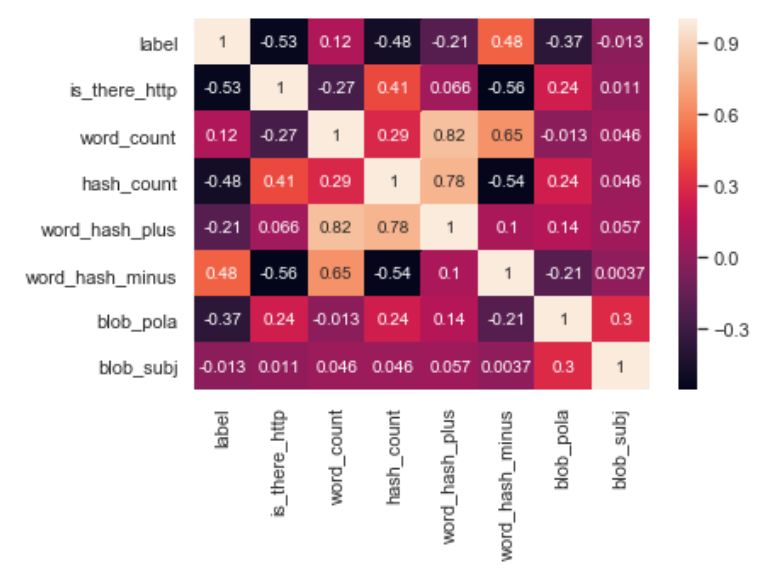
**Model Ensemble**In the previous section, various machine learning algorithms were used to train the tokenized tweets. In addition, the probability of predicting the class that each tweet from the test data can be determined for each of the model configuration. The figure below shows the probabilities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



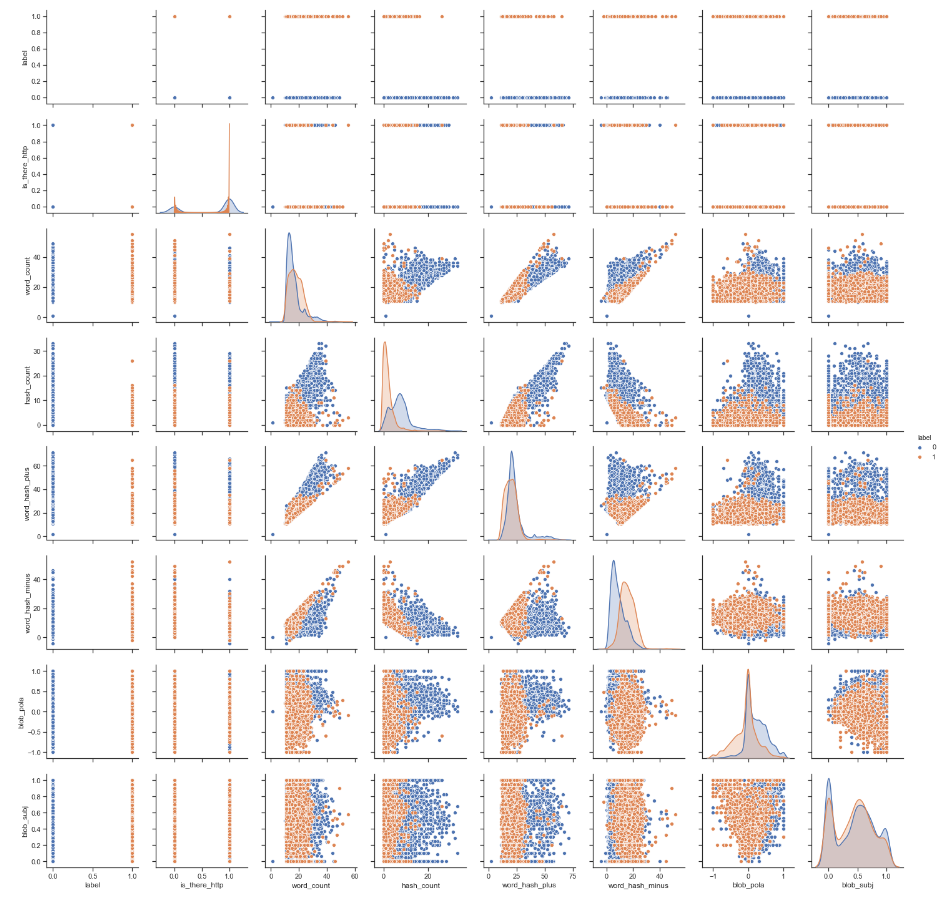
The difference in the predictions and the differences in the theory behind the different algorithms is a possible indication that one can check the linear combinations of the probabilities from the algorithms to see if the submitted could give a better F1-score. The algorithms considered are Multinomial 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 would improve but it did not. Finally, a majority vote base on the predicted class from each of the algorithm was considered if the weighted aggregate of the class predictions would lead to a better F1-score but it did not.

**Feature Selections Analysis**In this study, possible feature were identified and investigated. The number of words 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 was determined. 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 [2]) 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 see if the correct classes are predicted.

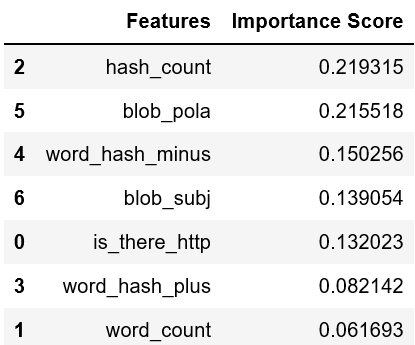
In order to check if any of the features may be redundant, the heat map of the feature correlation is determined. The figure below shows the correlation, the word\_count and the hash\_count are fairly correlated with the word\_hash\_plus. Since, the number of features is not many, one can still further investigate features that are relevant for predicting class label.



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. 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 one can do 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 used to tune a number of machine learning algorithms which were applied on the five selected features described above; the best parameters and the corresponding F1-scores are shown below.

|  |  |
| --- | --- |
| **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 |

**Conclusion**

**Future Work**Further analysis can be done on this data set. Consideration could be given to other natural language processing libraries like nltk, Spacy and Word2Vec. One could also use other analysis methodologies such as stemming and lemmatization which checks the root of each word. In addition, Word2Vec could be given consideration.

**Appendix**

1. **Confusion Matrix and associated metrics**

**References**

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

[2.] <https://textblob.readthedocs.io/en/dev/>

Viewed on April 22, 2019