Machine Learning Engineer Nanodegree
Capstone Project

Ashvin Srinivasan

May 20, 2018

1 Definition

With the advent of faster hardware and tremendous amount of data availability, a lot of analysis was possible from the past few years. Big data availed us to analyse the data and come up with models that could predict the desired information. To design these models, machine learning algorithms are used. There are many fields where these models could be employed, in general, wherever huge data is available. One such fields is opinion mining also known as sentiment analysis. "Sentiment analysis refers to the use of natural language processing, text analysis, computational linguistics to systematically identify, extract, quantify and study affective states and subjective information.[1]"

1.1 Project Overview

This project is on sentiment analysis. The main purpose of sentiment analysis is to classify a writer's attitude towards various topics into multiple categories; in our case it is positive and negative categories. Sentiment analysis has many applications in different domains including, but not limited to, business intelligence, politics, psychology among others. Recent years. With the explosive growth of social media such as blogs, micro-blogs, forum discussions, the web has drastically changed to the extent that nowadays billions of people all around the globe are freely allowed to conduct many activities such as interacting, sharing, posting contents resulting in a lot of data. Sentiment analysis is one class of computational techniques which automatically extracts and summarises the opinions of such immense volume of data which the average human reader is unable to process. This ocean of opinionated postings in social media is central to the individuals' activities as they impact our behaviours and help reshape businesses. Nowadays, not only individuals are no longer limited to asking friends and family about products but also businesses, organisations and companies do not require to conduct surveys or polls for opinions about products, as there are tons of user reviews and discussions in public forums on the Web. There are thus numerous immediate and practical applications and industrial interests of collecting and studying such opinions by using computational sentiment analysis techniques, spreading from consumer products, services, healthcare, and financial services to social events, political elections and more recently crisis management and natural disasters[2]. The research on sentiment analysis appeared even earlier than 2003 [3]. Research on sentiment analysis has been investigated in different perspectives. One of the most popular ways is to categorise these studies into three different levels, document level, sentence level and entity level.[4]

Document level The aim here is to analyse the overall sentiment of the entire document, for example given the review about a product, the task would be to determine whether it expresses positive or negative opinions about it.

Sentence level This level of analysis is limited to the sentences and their expressed opinions. For example, a review about a movie, specifically this level determines whether each sentence expresses a positive or a negative opinion

Entity level Instead of analysing it on document, sentence levels, this provides a much finer analysis for each aspect level or (feature based level).

1.2 Problem Statement

Our project addresses the sentence level based sentiment analysis, in which the model tries to categorise each statement into either positive or negative category. We try to solve this problem of classifying the statement by training a model on the training data set with the intention of minimising the loss function, thereby developing an accurate model. Concretely speaking, our datasets consists of two text files, each containing more than 5000 statements of positively and negatively classified statements. Our objective is to come up with model that can classify a given statement. Specifically, we address this problem statement by randomly shuffling the data sets and then dividing the data sets into training and test data sets. We train our model on the training data set and evaluate it on the test data set. It reduces to a supervised learning problem. We employ multiple algorithms in our model; Naive Bayes, Logistic regression, Linear Support Vector Classifier(SVC), Nu-SVC, Random Forest(RF) classifier. we optimise hyper parameters for each algorithm by a process of 1-fold validation. And finally, with the optimised models, we will have a majority voting based system from each of the five algorithmic based models that classifies a statement into a positive or negative category with a certain confidence level.

1.3 Metrics

We basically need metrics to quantify the quality of our model. There are three types of metrics used in our project:

1.3.1 Accuracy and F beta score

We shall be using two metrics to quantify our model; one is the accuracy and the other is f_β score. We quantify the solution through 'accuracy', which is mathematically defined as:

$$accuracy := sum(I)/n$$

where I is an indicator function=1, if $y = y_{pred}, else = 0$. y is the ground truth, y_{pred} is the predicted class from our trained model, and n is the total number of examples in the testing set. Accuracy is a simple yet effective metric to quantify our model's efficacy. Another metric that will be used is f_β score mathematically defined as:

$$f_\beta = (1 + \beta^2) * precision * recall / \Big((\beta^2 * precision) + recall \Big)$$

Where $Precision = TruePositives/(TruePositives + FalsePositives)$ and $Recall = TruePositives/(TruePositives + FalseNegatives)$. For our analysis, beta is typically set to '0.5'. An ideal value would be to have a F-score of 1. So the higher the value of F-score, the better is the model.

1.3.2 Mode and Confidence

Mode is defined as the most frequent value in the given set. In our case, we are using five algorithms. And each model outputs either 'Positive' or 'Negative'. The mode returns either 'Positive' or 'Negative' depending upon which occurs more frequently. We also use confidence that indicates the level of confidence with which the final model predicts the sentiment of the statement. Concretely speaking, if our model predicts an arbitrary statement as ['neg','neg','neg','neg','neg'] then the confidence level is 100% negative, where as if the prediction is ['neg','neg','neg','pos','pos'] then it is 60% negative.

Accuracy and F_β score are metrics to evaluate the quality of the model, where as mode and confidence are metrics with which the model classifies a given statement.

2 Analysis

2.1 Data Exploration and Exploratory Visualisation

The data used for sentiment analysis in this project is obtained from $'https://pythonprogramming.net/static/downloads/'$. It consists of two text files; one contains text that are all positively classified, while the other contains texts that are termed negative. Our analysis reduces to a supervised classification problem. The inputs are a set of several words as features while the target label assumes two values; 'Positive' or 'Negative'. We have 5300 texts each for positive and negative examples. We shall divide the total sets into training and testing sets, and use training set to train our model and test it out on our testing set.

Each statement will have words constituting different parts of speech Fig. 1. And these words will be used as input features for training our model. Each sentence has different parts of speech. There are a total of 8 parts of speech. A brief description about these parts of speech is a necessary condition to perform sentiment analysis. The different parts of speech are best explained with examples!

1) Noun: It typically is the name of a person, place, thing or an idea. My name is Ashvin, I received a letter from my teacher. Here Ashvin is a proper

Noun
Interjection
Pronoun
Parts of Speech
Preposition
Verb
Conjunction
Adverb
Adjective

Figure 1: Different Parts of Speech

noun, letter and teacher are nouns too. A noun generally can be a subject, indirect object[1]

2) Pronoun: It is a word used in place of a noun. For, eg. words like she, we, they, it and so on. Udacity's machine learning nano degree is awesome. It is a must for people learning ML. 'it' is a pronoun.

3) Verb: A verb expresses an action or being. For e.g., 'He brought me a macbook pro', here brought is a verb.

4) Adjective: It describes a noun or a pronoun. For. eg., 'Udacity is an awesome website, it provides an amazing ML nanodegree'. Awesome and amazing are adjectives that describes something about Udacity and MLND.

5) Adverb: It describes a verb. For e.g., Udacity reviewers/mentors swiftly respond to your queries. Swiftly is an adverb.

6) Preposition: it is a word placed before a noun/pronoun to form a phrase modifying another word in the sentence. Words like 'by', 'with', 'about', 'until' are prepositions.

7) Conjunction: A conjunction joins words, phrases and clauses. For e.g., words like 'and', 'but', 'or'.

8) Interjection: An interjection is a word used to express emotion. For e.g., 'Oh!', 'Wow', 'Oops!'

These are the parts of speech typically used in the English language. Understanding the parts of speech helps us to narrow down our features. There is direct relation between the parts of speech and sentiment of a sentence. For eg, an adjective, adverb describes the quality of a noun, verb and thereby contributing more to the sentiments as compared to maybe a preposition. Our data set contains such sentences, and it is our model's job to accurately classify them. Thus, our datasets basically contains words from sentences as our input features and target label is a categorical variable; 'positive' or 'negative' that describes the sentiment of our text, which directly falls into the sentiment analysis paradigm.

2.2 Algorithms and Techniques

For this project, we use a total of five algorithms; naive Bayes, logistic regression, linear SVC, Nu-SVC, Random forest classifiers. Since our problem is reduced to a supervised classification problem, the aforementioned algorithms are all classification based algorithms.

Original naive Bayes model works on the assumption that every feature is independent of each other, i.e., every word is independent of every other word. Hence the name Naive Bayes, it works on Bayes theorem. Though this assumption is not valid, it works fairly good owing to the shear number of features(words).

Logistic regression algorithm is a simple but effective classification algorithm, we use cross entropy as the cost function that needs to be minimised which results in a better model.

The third algorithm used is linear SVC, it works on the principle of maximising the margin, we are using a linear kernel, quite similar to the logistic regression but here the separating hyperplane is selected in a such a way that the margins are maximised. C is a hyper parameter that we optimise, it is closely related to the regularisation parameter. We need to optimise C to strike a balance between over fitting(variance) and under fitting(bias).

The fourth algorithm we use is Nu-SVC, the kernel used here is 'rbf'(radial basis function), this kernel transforms the data into a higher dimensional space where the data tends to be linearly separable. The hyper parameters used here are γ; to control the width of the Gaussian kernel, and 'Nu'; an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1].

The final algorithm incorporated is random forest. It is an ensemble of multiple decision trees. In each decision tree, split occurs at each node for a certain feature. Each feature is selected such that there is maximum decrease in entropy or maximum increase in information gain. The hyper parameters used are 'n-estimators', the number of decision trees. 'criterion'; the function to measure the quality of a split. Supported criteria are 'gini' for the Gini impurity and 'entropy' for the information gain

Having optimised the above algorithms, we classify a statement as positive or negative based on the majority vote. And we also calculate the level of confidence with which prediction is made!

2.3 Benchmark Model

A naive predictor is a good benchmark model that relates to our problem statement. In order to compare our results obtained from training, we can stack it up against a naive predictor. When we have a model that always predicts 'positive' (i.e. the text conveys a positive sentiment) then our model will have no True Negatives(TN) or False Negatives(FN) as we are not making any negative('negative') predictions. Therefore our Accuracy in this case becomes the same as our Precision(True Positives/(True Positives + False Positives)) as every prediction that we have made with 'positive' that should have 'negative' becomes a False Positive; therefore our denominator in this case is the total number of records we have in total[2]. The naive predictor simply predicts a constant value without taking any input features into account, hence the name naive predictor. Mathematically it is given as:

$$accuracy_{naive} = TruePositives/(TruePositives + FalsePositives)$$

By obtaining this naive accuracy, we could then stack it up against the accuracy obtained from our trained model and thereby quantifying our model's performance.

3 Methodology

3.1 Data Preprocessing and Impementation

Each statement contains different parts of speech. Considering that we have about 10,000 plus statements, and every statement can have as many words. Each word is to be considered as a feature. The issue with this approach is that the total number of features is as wide as the number of words in the English dictionary. And thus, using all these available features would tremendously increase the input dimensional space, which is not feasible as this could lead to overfitting of the data. One of the preprocessing techniques is to calculate the frequency distribution of the words occurring in our dataset and considering only the top 5000 words as features.

3.1.1 Top words Extraction

The code used for extracting the top 5000 words is

```
# pos_reviews_words have words  from positive
# text and likewise for neg_reviews_words

pos_reviews_words=nltk.word_tokenize(pos_reviews)
neg_reviews_words=nltk.word_tokenize(neg_reviews)
imp_pos_words=extract(pos_reviews_words)
imp_neg_words=extract(neg_reviews_words)

# all_words contain the set of words
#occurring in all documents

all_words=[]
for ii in imp_pos_words:  # put pos_reviews_words to include all words
    all_words.append(ii.lower())
for ii in imp_neg_words:  # put neg_reviews_words to include all words
    all_words.append(ii.lower())

# all_words_freq contains all words
# its associated frequencies
all_words_freq=nltk.FreqDist(all_words)
word_feats=list(all_words_freq.keys())[:5000]
```

We use the NLTK package to first tokenize the statements and extract the words from each statement. Upon extracting the words, we extract its associated parts of speech. So the list *all_words* have the words and its tagged parts of speech. Once we have that, we calculate the frequency distribution and store it in *words_feats*.

3.1.2 Extraction of Adjectives, Verbs, Nouns

Data Exploration clearly explained the different parts of speech. As we can see that the sentiment often is reflected more in certain parts of speech among others. In this regard, we consider "Adjective", "Verb", and "Noun" as they are more likely to contribute to the sentiment. In order to extract this we use the *extract* function in which we declare a list *all_words_types* that contains the list of parts of speech that we need to extract, for e.g., $'J','V','N'$ stand for Adjective, Verb, and Noun.

```
def extract(word_list):
    all_word_types=['J','V','N']
    ret_list=[]
    temp=nltk.pos_tag(word_list)
    for w in temp:
        if (w[1][0]) in all_word_types:
            ret_list.append(w[0].lower())

    return ret_list
```

3.1.3 Algorithms, Metrics

One of the major hurdles we faced during the course of this project implementation were the computation time for features extraction, training model, hyper parameters optimisation. To circumvent this problem, we made use of the pickle package in python, in which we can save the already trained models, optimised parameters, and extracted features. As a result of this, we don't have to retrain the model over and over. An obvious outcome of this is that we have our main codes in the form of *sentiment_analysis_train.ipynb* and *sentiment_analysis_predict.ipynb*. Initially we have been provided only with the two data sets containing the positively and negatively classified texts. From these datasets we divide it into training and testing data sets. We use the previously mentioned functions to extract the wanted data. Once extracted, we call the $find\_feats$ function that basically assigns a boolean value; True or False for all the 5000 features indicating us which of these 5000 words are present in a particular statement.

```
def find_feats(document):
    words=nltk.word_tokenize(document)
    words=extract(words)
    features={}
    for w in word_feats:
          features[w]=w in words
    return features
```

Once we have extracted the features for all the texts, we are ready to train our model. Our model, like previously mentioned, consists of five algorithms. Logistic Regression, Original NB, linear-SVC, Nu-SVC, and RF classifier. For all the classifiers, we use two metrics that we import from sklearn package in python. They are accuracy and fbeta scores.

Original NB The NB algorithm is an intuitive method that uses probabilities of each feature belonging to each class to make a prediction. NB simplifies thee calculation of probabilities by assuming that the probability of each attribute belonging to a given class value is independent of all other attributes. This is a strong assumption but results in a fast and effective method. The probability of a class value given a value of an attribute is called the conditional probability. By multiplying the conditional probabilities together for each attribute for a given class value, we have a probability of a data instance belonging to that class. To make a prediction we can calculate probabilities of the instance belonging to each class and select the class value with the highest probability. We use the following code implementation:

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
acc=(nltk.classify.accuracy(classifier, test_set))*100
```

Logistic regression Logistic regression is a statistical method for analysing a dataset. It is much similar to linear regression in that we come with a fitting line that tries to define the relationship between our features(words) with the output variable(positive or negative). While training the model, we come up with parameters that define the model

```
LogisticRegression_classifier.train(train_set)
acc,beta=find_pred(LogisticRegression_classifier,test_set)
```

Linear SVC This is very similar to logistic classifier. However, this algorithm comes up with a classification boundary such that it maximises the margin. The kernel used here is a linear kernel. We implement this by the following code.

```
unopt_linSVC_classifier.train(train_set)
unopt_acc,unopt_beta=find_pred(unopt_linSVC_classifier,test_set)
```

Nu-SVC Much similar like the linear SVC, the only difference is that we use a non linear kernel such as 'rbf' or a 'poly' kernel. It is more effective when the data is not linear separable. The usage of non linear kernel transforms the data into a higher dimensional space where the data eventually becomes a linearly separable. For e.g., a RBF or Gaussian kernel, effectively transforms the data into an infinite dimensional space.

```
unopt_NuSVC_classifier.train(train_set)
unopt_acc,unopt_beta=find_pred(unopt_NuSVC_classifier,test_set)
```

Random Forest This is an ensemble technique using multiple decision trees. Using multiple trees makes this model more robust, less susceptible to variance in data.

```
unopt_rf_classifier.train(train_set)
unopt_acc,unopt_beta=find_pred(unopt_rf_classifier,test_set)
```

As we can see the above simple code implementations for all the above algorithms, it has a similar approach, first we train on the training dataset and predict it on the testing dataset. We use the *find_pred* function for prediction.

```
def find_pred(clf,test_set):
    y_pred=[clf.classify(test_set[i][0]) for i in range(len(test_set))]
    y_true=[test_set[i][1] for i in range(len(test_set))]
    y_pred=pd.Series(y_pred).apply(lambda x:0 if x=='neg' else 1)
    y_true=pd.Series(y_true).apply(lambda x:0 if x=='neg' else 1)
    accuracy=accuracy_score(y_true,y_pred)*100
    f_beta=fbeta_score(y_true,y_pred,beta=0.5)
    return accuracy,f_beta
```

3.2 Refinement

Three out of five algorithms have hyper parameters that needs to be optimised. Optimising these parameters is likely to increase the accuracy of these models. Initial unoptimised results and optimised results are reported in the following section. We optimise the 'C' parameter. This term is closely related to the regularisation term. This parameter is used to strike a balance between over fitting and under fitting. A rough indication is, higher the value C, it reduces bias and lesser it is it reduces variance in the data. The different values used are $C = [2^-3, 2^-1, 2^0, 2^2, 2^4, 2^6, 2^8, 2^{10}]$.

For Nu-SVC we use two hyper parameters 1) $nu = [0.2, 0.4, 0.6, 0.8]$ 2) $'kernel' = ['rbf', 'poly']$, nu specifies an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval $(0, 1]$. Kernel parameter is used to specify the type of the kernel to be used.

The final algorithm is random forest classifier, which has three hyper parameters we optimise; $n\_estimators = [5, 10, 15]$, this specifies the number of decision trees to be used in a random forest model. The second parameter $min\_samples\_split' = [2, 10, 50]$, this indicates the minimum number of samples in the final node below which no more splits would occur. This is an important parameter to gauge the effectiveness of the model in terms of overfitting the model. The third parameter we optimise is criterion which we have set it to only $'entropy'$. we use these models for final training of the data. To obtain the hyper parameters we further divide the training set into training set and validation set. We use leave one out cross validation to obtain the accuracy, fbeta scores and thereby optimise the hyper parameters. Upon finding the best hyper parameters, we combine our final algorithms and develop a type of majority voting system. This type of majority voting systems is better than using a single predictive algorithm. It also reduces the inherent variance of the model. We can thus predict the sentiment with more confidence and the following code implementation is as follows!

```
class VoteClassifier(ClassifierI):
    def __init__(self, *classifiers):
        self._classifiers = classifiers

    def classify(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)
        return votes

    def confidence(self, features):
        votes = []
        for c in self._classifiers:
            v = c.classify(features)
            votes.append(v)

        choice_votes = votes.count(mode(votes))
        conf = choice_votes / len(votes)
        return conf
```

4 Results

4.1 Model Evaluation and Validation

Our final model was chosen as an ensemble of five algorithms as it increases the robustness of our model. We individually optimise every model based on validation, upon which we use a vote classifier that combines the result of multiple models as opposed to a single one, which also reduce any inherent bias of any single model. This type of ensemble ensures more reliability in our results which is reflected in the confidence level.

The final results are documented in the following table. The pre- and post- optimised accuracies and fbeta scores are shown.

Algorithm	Accuracy	Improved Accuracy	f_β score	Improved f_β score
Naive Bayes	66.7	-		
Logistic classifier	66.2	-	0.675	-
Linear SVC	63.7	64.7	0.644	0.653
Nu-SVC	59.48	66.41	0.60	0.666
Random Forest	60.6	62.6	0.618	0.634

As can be seen from the table all the algorithms have an improved accuracy and fbeta scores as compared to unoptimised scores. This justifies the use of optimised models

4.2 Justification

Our final model that incorporates the optimised algorithms, sees an increase in accuracy and fbeta scores. Which is shown in the following table. We see that there is an increase of more than 15 percent in accuracy and also a considerable increase fbeta score as well

Model	Accuracy	f_β score
Optimised model	66.41	0.666
Bench Mark model	49.9	0.55

We further justify our model by testing it out on random tweets extracted from Twitter. These tweets are extracted from an environment totally different from our datasets. This should be a good indication about the robustness of our model. The following tweets are extracted from twitter using tweepy package, once we get the access tokens and consumer keys, we use a script to extract the tweets. The keyword we searched for was 'cricket'. Below are some of the mentioned extracted tweets and its associated classified categories with confidence levels. Please see the attached file 'final_tweet_analysis.csv' for more details.

'Cricket is horrible' classified as negative with 100% confidence

'Just a perfect day for picnic, beers and Middlesex v Kent in RL50 county cricket?' classified as positive with 100% confidence

'Hey Cricket fans be ready for today's game. As it is very crucial game for ThumsUpCharged ThumsUpForMI' classified as negative with 80% confidence.

'Virat Kohli breaks silence on fitness concerns, promises answers during CSK v/s KXIP match - Daily News' classified as positive with 100% confidence

5 Conclusion

We have been successful in extracting certain features(words), using them in developing our models. The initial benchmark model reported an accuracy of 50% owing to the fact that there are about equal amount of statements belonging to each category. Upon developing our models we saw an increase in more than 15% accuracy as shown in the following diagrams. This strongly reinstates our faith in the model as we get a better accuracy.

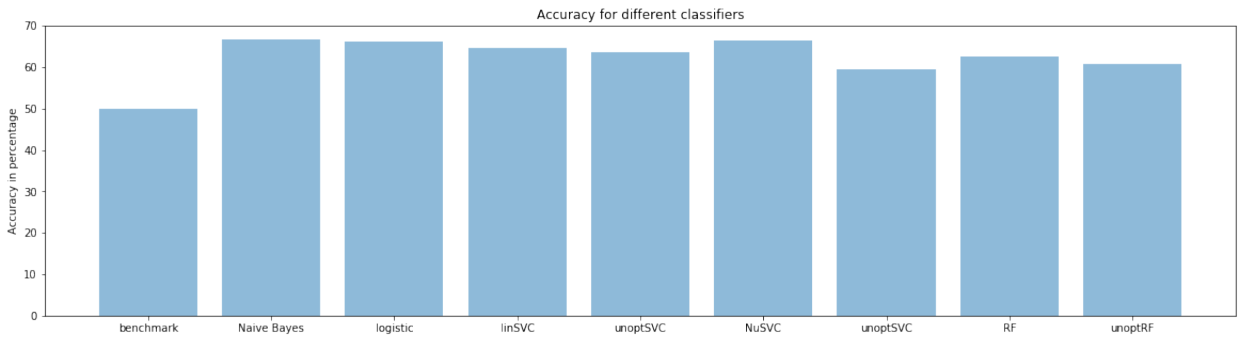

Figure 2: accuracy for different classifiers

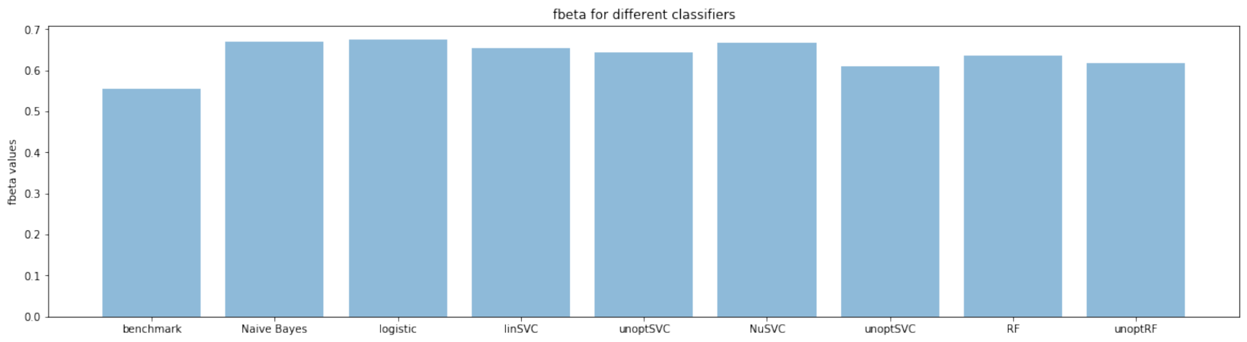

Figure 3: fbeta for different classifiers

We can also see from fig. 4 that around 60% of the statements are classified with a confidence level of 100% which suggests about the good reliability of our model.

Looking back at this project, from the phase of data extraction, data preparation, and model optimisation, there was a lot of learning involved. The major hurdles was about feature selection. Since sentiment analysis involved using words as features, and the shear amount of words in the English language puts a high necessity for good, meaningful features. We addressed this issue by considering the most frequently occurring words and in that we also extracted only those parts of speech which would be more impactful in determining the sentiment. The other hurdle was eliminating the

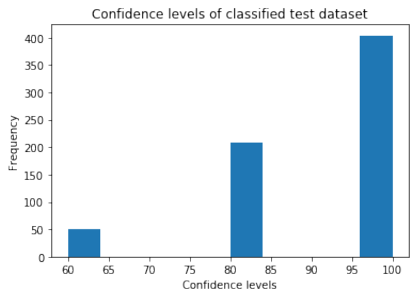

Figure 4: Confidence levels for classified test dataset

inherent bias present in a given model, which we addressed by optimising the model and using multiple models. Finally, to test out our model we extracted totally random statements from Twitter for analysis and found that our model's performance was very reliable. Further improvements could likely include using other algorithms in our model and also have more categories rather than just 'positive' or 'negative'.

6 References

[1] https://en.wikipedia.org/wiki/Sentiment_analysis

[2] http://www.public.asu.edu/ gbeigi/files/BeigiSentimentChapter.pdf

[3] Sanjiv Das and Mike Chen. Yahoo! for amazon: Extracting market sentiment from stock message boards. In Asia Pacific Finance Association Annual Conf. (APFA) , 2001.