Major Project
Twitter Sentiment Analysis

Piyush Nautiyal
(24/AI/M1216)

Abstract

In this report, I address the problem of sentiment classification on twitter dataset. I use a number of machine learning and deep learning methods to perform sentiment analysis. In the end of Problem Statement I get a accuracy of 83.34%

Twitter is a popular social networking website where members create and interact with messages known as "tweets". This serves as a mean for individuals to express their thoughts or feelings about different subjects. Various different tweets such as consumers and marketers have done sentiment analysis on such tweets to gather insights into products or to conduct market analysis. Furthermore, with the recent advancements in machine learning algorithms, I are able improve the accuracy of My sentiment analysis predictions.

In this report, I will attempt to conduct sentiment analysis on "tweets" using various different machine learning algorithms. I attempt to classify the polarity of the tweets where it is either positive or negative. If the tweets has both positive and negative elements, the more dominant sentiment should be picked as the final label.

I use the dataset from Kaggle which was crawled and labeled positive/negative. The data provided comes with emoticons, usernames and hashtags which are required to be processed and converted into a standard form. I also need to extract useful features from the text such unigrams and bigrams which is a form of representation of the "tweets"..

1 Data Description

The data given is in the form of a comma-separated values files with tweets and their corresponding sentiments. The training dataset is a csv file of type tweetst_id,sentiment,tweetst where the tweetst_id is a unique integer identifying the tweets, sentiment is either 1 (positive) or 0 (negative), and tweetst is the tweetst enclosed in "". Similarly, the test dataset is a csv file of type tweetst_id, tweetst.

The dataset is a mixture of words, emoticons, symbols, URLs and references to people. Words

	Total	Unique	Average	Max	Positive	Negative
Tweets	800000	-	-	-	400312	399688
User Mentions	393392	-	0.4917	12	-	-
Emoticons	6797	-	0.0085	5	5807	990
URLs	38698	-	0.0484	5	-	-
Unigrams	9823554	181232	12.279	40	-	-
Bigrams	9025707	1954953	11.28	-	-	-

Table 1: Statistics of preprocessed train dataset

	Total	Unique	Average	Max	Positive	Negative
Tweets	200000	-	-	-	-	-
User Mentions	97887	-	0.4894	11	-	-
Emoticons	1700	-	0.0085	10	1472	228
URLs	9553	-	0.0478	5	-	-

Unigrams	2457216	78282	12.286	36	-	-
Bigrams	2257751	686530	11.29	-	-	-

Table 2: Statistics of preprocessed test dataset

and emoticons contribute to predicting the sentiment, but URLs and references to people don't. Therefore, URLs and references can be ignored. The words are also a mixture of misspelled words, extra punctuations, and words with many repeated letters. The tweets, therefore, have to be preprocessed to standardize the dataset.

The provided training and test dataset have 800000 and 200000 tweets respectively. Preliminary statistical analysis of the contents of datasets, after preprocessing as described in section 3.1, is shown in tables 1 and 2.

2 Methodology and Implementation

2.1 Pre-processing

Raw tweets scraped from twitter generally result in a noisy dataset. This is due to the casual nature of people's usage of social media. Tweets have certain special characteristics such as retweets, emoticons, user mentions, etc. which have to be suitably extracted. Therefore, raw twitter data has to be normalized to create a dataset which can be easily learned by various classifiers. I have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. I first do some general pre-processing on tweets which is as follows.

- Convert the tweetst to lower case.
- Replace 2 or more dots (.) with space.
- Strip spaces and quotes (" and') from the ends of tweetst.
- Replace 2 or more spaces with a single space.

I handle special twitter features as follows.

2.1.1 URL

Users often share hyperlinks to other webpages in their tweets. Any particular URL is not important for text classification as it would lead to very sparse features. Therefore, I replace all the URLs in tweets with the word URL. The regular expression used to match URLs is ((www\.[\S]+)|(https?://[\S]+)).

2.1.2 User Mention

Every twitter user has a handle associated with them. Users often mention other users in their tweets by @handle. I replace all user mentions with the word USER_MENTION. The regular expression used to match user mention is @[\S]+.

<table>
<tr><th>Emoticon(s)</th><th>Type</th><th>Regex</th><th>Replacement</th></tr>
<tr><td>:), :), :-), (:, (:, (-:, :')</td><td>Smile</td><td>(:\s?\)|:-\)|\(\s?:|\(-:|:\'\))</td><td>EMO_POS</td></tr>
<tr><td>:D, : D, :-D, xD, x-D, XD, X-D</td><td>Laugh</td><td>(:\s?D|:-D|x-?D|X-?D)</td><td>EMO_POS</td></tr>
<tr><td>;-), ;), ;-D, ;D, (;, (-;</td><td>Wink</td><td>(:\s?\(|:-\(|\)\s?:|\)-:)</td><td>EMO_POS</td></tr>
<tr><td><3, :*</td><td>Love</td><td>(<3|:*)</td><td>EMO_POS</td></tr>
<tr><td>:-(, : (, :(,):,)-:</td><td>Sad</td><td>(:\s?\(|:-\(|\)\s?:|\)-:)</td><td>EMO_NEG</td></tr>
<tr><td>:,(, :'(, :"(</td><td>Cry</td><td>(:,\(|:\'\(|:"\()</td><td>EMO_NEG</td></tr>
</table>

Table 3: List of emoticons matched by My method

2.1.3 Emoticon

Users often use a number of different emoticons in their tweetst to convey different emotions. It is impossible to exhaustively match all the different emoticons used on social media as the number is ever increasing. However, I match some common emoticons which are used very frequently. I replace the matched emoticons with either EMO_POS or EMO_NEG depending on whether it is conveying a positive or a negative emotion. A list of all emoticons matched by My method is given in table 3.

2.1.4 Hashtag

Hashtags are unspaced phrases prefixed by the hash symbol (#) which is frequently used by users to mention a trending topic on twitter. I replace all the hashtags with the words with the hash symbol. For example, #hello is replaced by hello. The regular expression used to match hashtags is #(\S+).

2.1.5 Retweets

Retweets are tweets which have already been sent by someone else and are shared by other users. Retweets begin with the letters RT. I remove RT from the tweets as it is not an important feature for text classification. The regular expression used to match retweets is \brt\b.

After applying tweetst level pre-processing, I processed individual words of tweets as follows.

- Strip any punctuation ['"?!,.():;] from the word.
- Convert 2 or more letter repetitions to 2 letters. Some people send tweets like *I am sooooo happppppy* adding multiple characters to emphasize on certain words. This is done to handle such tweets by converting them to *I am soo happy*.
- Remove - and '. This is done to handle words like t-shirt and their's by converting them to the more general form tshirt and theirs.
- Check if the word is valid and accept it only if it is. I define a valid word as a word which begins with an alphabet with successive characters being alphabets, numbers or one of dot (.) and underscore(_).

Some example tweets from the training dataset and their normalized versions are shown in table 4.

2.2 Feature Extraction

I extract two types of features from My dataset, namely unigrams and bigrams. I create a frequency distribution of the unigrams and bigrams present in the dataset and choose top N unigrams and bigrams for My analysis.

2.2.1 Unigrams

Probably the simplest and the most commonly used features for text classification is the presence of single words or tokens in the the text. I extract single words from the training dataset and create a frequency distribution of these words. A total of 181232 unique words are extracted from Raw misses Swimming Class. http://plurk.com/p/12nt0b

Normalized	misses swimming class URL
Raw	@98PXYRochester HEYYYYYYYYY!! its Fer from Chile again
Normalized	USER_MENTION heyy its fer from chile again
Raw	Sometimes, You gotta hate #Windows updates.
Normalized	sometimes you gotta hate windows updates
Raw	@Santiago_Steph hii come talk to me i got candy :)
Normalized	USER_MENTION hii come talk to me i got candy EMO_POS

Raw	@bolly47 oh no :'(r.i.p. yMy bella
Normalized	USER_MENTION oh no EMO_NEG r.i.p yMy bella

Table 4: Example tweets from the dataset and their normalized versions.

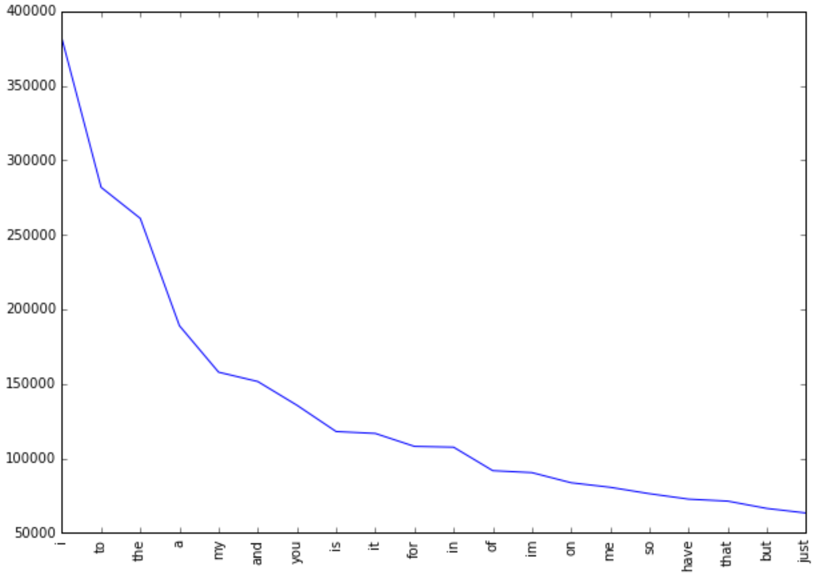

Figure 1: Frequencies of top 20 unigrams.

the dataset. Out of these words, most of the words at end of frequency spectrum are noise and occur very few times to influence classification. I, therefore, only use top N words from these to create My vocabulary where N is 15000 for sparse vector classification and 90000 for dense vector classification. The frequency distribution of top 20 words in My vocabulary is shown in figure 1. I can observe in figure 2 that the frequency distribution follows Zipf's law which states that in a large sample of words, the frequency of a word is inversely proportional to its rank in the frequency table. This can be seen by the fact that a linear trendline with a negative slope fits the plot of $\log(Frequency)$ vs. $\log(Rank)$. The equation of the trendline shown in figure 2 is $\log(Frequency) = -0.78\log(Rank) + 13.31$.

2.2.2 Bigrams

Bigrams are word pairs in the dataset which occur in succession in the corpus. These features are a good way to model negation in natural language like in the phrase – *This is not good.* A total of

1954953 unique bigrams Ire extracted from the dataset. Out of these, most of the bigrams at end of frequency spectrum are noise and occur very few times to influence classification. I therefore use only top 10000 bigrams from these to create My vocabulary. The frequency distribution of top 20 bigrams in My vocabulary is shown in figure 3.

2.3 Feature Representation

After extracting the unigrams and bigrams, I represent each tweetst as a feature vector in either sparse vector representation or dense vector representation depending on the classification method.

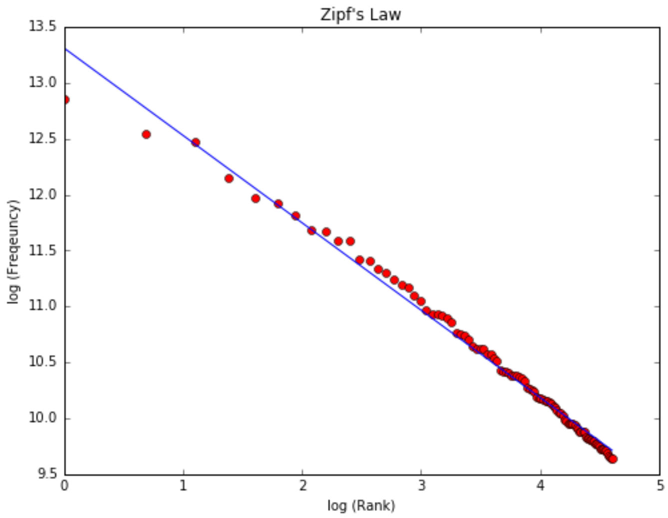

Figure 2: Unigrams frequencies follow Zipf's Law.

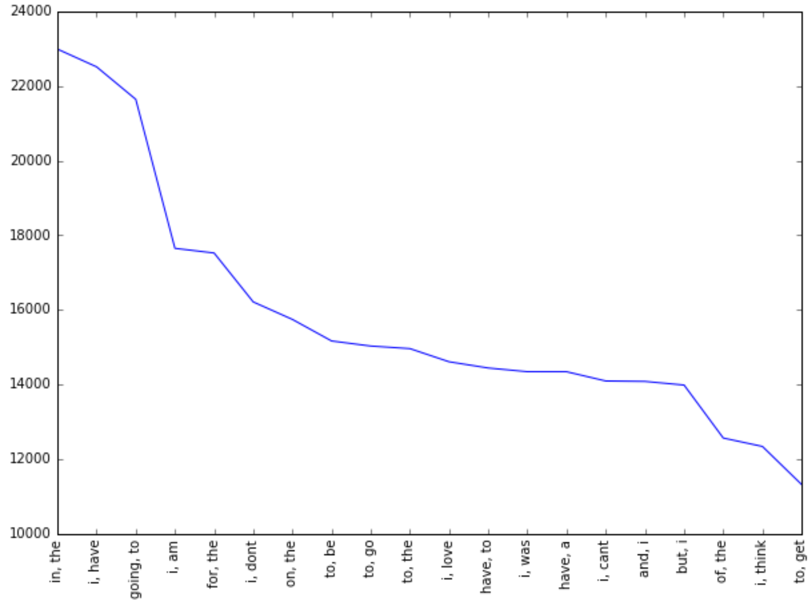

Figure 3: Frequencies of top 20 bigrams.

2.3.1 Sparse Vector Representation

Depending on whether or not I are using bigram features, the sparse vector representation of each tweetst is either of length 15000 (when considering only unigrams) or 25000 (when considering unigrams and bigrams). Each unigram (and bigram) is given a unique index depending on its rank. The feature vector for a tweetst has a positive value at the indices of unigrams (and bigrams) which are present in that tweetst and zero elsewhere which is why the vector is sparse. The positive value at the indices of unigrams (and bigrams) depends on the feature type I specify which is one of *presence* and *frequency*.

- *presence* In the case of *presence* feature type, the feature vector has a 1 at indices of unigrams (and bigrams) present in a tweetst and 0 elsewhere.
- *frequency* In the case of *frequency* feature type, the feature vector has a positive integer at indices of unigrams (and bigrams) which is the frequency of that unigram (or bigram) in the tweetst and 0 elsewhere. A matrix of such term-frequency vectors is constructed for the entire training dataset and then each term frequency is scaled by the inverse-document-frequency of the term (idf) to assign higher values to important terms. The inverse-document-frequency of a term t is defined as.

  $$idf(t) = \log\left(\frac{1+n_d}{1+df(d,t)}\right) + 1$$

  where n_d is the total number of documents and $df(d,t)$ is the number of documents in which the term t occurs.

Handling Memory Issues Which dealing with sparse vector representations, the feature vector for each tweetst is of length 25000 and the total number of tweets in the training set is 800000 which means allocation of memory for a matrix of size 800000×25000. Assuming 4 bytes are required to represent each float value in the matrix, this martix needs a memory of 8×10^{10} bytes (≈ 75 GB) which is far greater than the memory available in common notebooks. To tackle this issue, I used scipy.sparse.lil_matrix data structure provided by Scipy which is a memory efficient linked list based implementation of sparse matrices. In addition to that, I used Python generators wherever possible instead of keeping the entire dataset in memory.

2.3.2 Dense Vector Representation

For dense vector representation I use a vocabulary of unigrams of size 90000 i.e. the top 90000 words in the dataset. I assign an integer index to each word depending on its rank (starting from 1) which means that the most common word is assigned the number 1, the second most common word is assigned the number 2 and so on. Each tweetst is then represented by a vector of these indices which is a dense vector.

2.4 Classifiers

2.4.1 Naive Bayes

Naive Bayes is a simple model which can be used for text classification. In this model, the class $\hat{c}$ is assigned to a tweetst t, where

$$\hat{c} = \underset{c}{argmax}\, P(c|t)$$

$$P(c|t) \propto P(c)\prod_{i=1}^{n} P(f_i|c)$$

In the formula above, f_i represents the i-th feature of total n features. P(c) and $P(f_i|c)$ can be obtained through maximum likelihood estimates.

2.4.2 Maximum Entropy

Maximum Entropy Classifier model is based on the Principle of Maximum Entropy. The main idea behind it is to choose the most uniform probabilistic model that maximizes the entropy, with given constraints. Unlike Naive Bayes, it does not assume that features are conditionally independent of each other. So, I can add features like bigrams without worrying about feature overlap. In a binary classification problem like the one I are addressing, it is the same as using Logistic Regression to find a distribution over the classes. The model is represented by

$$P_{ME}(c|d,\lambda) = \frac{exp[\sum_i \lambda_i f_i(c,d)]}{\sum_{c'} exp[\sum_i \lambda_i f_i(c,d)]}$$

Here, c is the class, d is the tweetst and λ is the Iight vector. The Iight vector is found by numerical optimization of the lambdas so as to maximize the conditional probability.

2.4.3 Decision Tree

Decision trees are a classifier model in which each node of the tree represents a test on the attribute of the data set, and its children represent the outcomes. The leaf nodes represents the final classes of the data points. It is a supervised classifier model which uses data with known labels to form the decision tree and then the model is applied on the test data. For each node in the tree the best test condition or decision has to be taken. I use the GINI factor to decide the best split. For a given node t, $GINI(t) = 1 - \sum_j [p(j|t)]^2$, where $p(j|t)$ is the relative frequency of class j at node t, and $GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$ (n_i = number of records at child i, n = number of records at node p)indicates the quality of the split. I choose a split that minimizes the GINI factor.

2.4.4 Random Forest

Random Forest is an ensemble learning algorithm for classification and regression. Random Forest generates a multitude of decision trees classifies based on the aggregated decision of those trees. For a set of tweets $x_1, x_2, \ldots x_n$ and their respective sentiment labels $y_1, y_2, \ldots_n$ bagging repeatedly selects a random sample (X_b, Y_b) with replacement. Each classification tree f_b is trained using a different random sample (X_b, Y_b) where b ranges from $1 \ldots B$. Finally, a majority vote is taken of predictions of these B trees.

2.4.5 XGBoost

Xgboost is a form of gradient boosting algorithm which produces a prediction model that is an ensemble of Iak prediction decision trees. I use the ensemble of K models by adding their outputs in the following manner

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

where F is the space of trees, x_i is the input and $\hat{y}_i$ is the final output. I attempt to minimize the following loss function

$$L(\Phi) = \sum_i l(\hat{y}_i, y_i) + \sum \Omega(f_k)$$

$$where\ \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

where Ω is the regularisation term.

2.4.6 SVM

SVM, also known as support vector machines, is a non-probabilistic binary linear classifier. For a training set of points (x_i, y_i) where x is the feature vector and y is the class, I want to find the maximum-margin hyperplane that divides the points with $y_i = 1$ and $y_i = -1$. The equation of the hyperplane is as follow

$$w \cdot x - b = 0$$

I want to maximize the margin, denoted by γ, as follows

$$\max_{w,\gamma} \gamma, s.t. \forall i, \gamma \leq y_i(w \cdot x_i + b)$$

in order to separate the points Ill.

2.4.7 Multi-Layer Perceptron

MLP or Multilayer perceptron is a class of feed-forward neural networks, which has atleast three layers of neurons. Each neuron uses a non-linear activation function, and learns with supervision using backpropagation algorithm. It performs Ill in complex classification problems such as sentiment analysis by learning non-linear models.

2.4.8 Convolutional Neural Networks

Convolutional Neural Networks or CNNs are a type of neural networks which involve layers called convolution layers which can interpret spacial data. A convolution layers has a number of filters or kernels which it learns to extract specific types of features from the data. The kernel is a 2D window which is slided over the input data performing the convolution operation. I use temporal convolution in My experiments which is suitable for analyzing sequential data like tweets.

2.4.9 Recurrent Neural Networks

Recurrent Neural Network are a network of neuron-like nodes, each with a directed (one-way) connection to every other node. In RNN, hidden state denoted by h_t acts as memory of the network and learns contextual information which is important for classification of natural language. The output at each step is calculated based on the memory h_t at time t and current input x_t. The main feature of an RNN is its hidden state, which captures sequential dependence in information. I used Long Term Short Memory (LSTM) networks in My experiments which is a special kind of RNN capable of remembering information over a long period of time.

3 Experiments

I perform experiments using various different classifiers. Unless otherwise specified, I use 10% of the training dataset for validation of My models to check against overfitting i.e.

For Naive Bayes, Maximum Entropy, Decision Tree, Random Forest, XGBoost, SVM and Multi-Layer Perceptron I use sparse vector representation of tweets. Baseline

For a baseline, I use a simple positive and negative word counting method to assign sentiment to a given tweetst. I use the Opinion Dataset of positive and negative words to classify tweets. In cases when the number of positive and negative words are equal, I assign positive sentiment. Using this baseline model, I achieve a classification accuracy of 19.30

3.1 Naive Bayes

I used MultinomialNB from sklearn.naive_bayes package of *scikit-learn* for Naive Bayes classification. I used Laplace smoothed version of Naive Bayes with the smoothing parameter α set to its default value of 1. I used sparse vector representation for classification and ran experiments using both *presence* and *frequency* feature types. I found that *presence* features outperform *frequency* features because Naive Bayes is essentially built to work better on integer features rather than floats. I also observed that addition of bigram features improves the accuracy. I obtain a best validation accuracy of 79.68% using Naive Bayes with *presence* of unigrams and bigrams. A comparison of accuracies obtained on the validation set using different features is shown in table
5.

3.2 Maximum Entropy

The nltk library provides several text analysis tools. I use the Max ent Classifier to perform sentiment analysis on the given tweets. Unigrams, bigrams and a combination of both Ire given as input features to the classifier. The Improved Iterative Scaling algorithm for training provided better results than Generalised Iterative Scaling. Feature combination of unigrams and bigrams, gave better accuracy of 80.98% compared to just unigrams (79.34%) and just bigrams (79.2%).

For a binary classification problem, Logistic Regression is essentially the same as Maximum Entropy. So, I implemented a sequential Logistic Regression model using keras, with sigmoid activation function, binary cross-entropy loss and Adam's optimizer achieving better performance than nltk. Using frequency and presence features I get almost the same accuracies, but the performance is slightly better when I use unigrams and bigrams together. The best accuracy achieved was 81.52%. A comparison of accuracies obtained on the validation set using different features is shown in table 5.

3.3 Decision Tree

I use the DecisionTreeClassifier from sklearn.tree package provided by *scikit-learn* to build My model. GINI is used to evaluate the split at every node and the best split is chosen always. The model performed slightly better using the presence feature compared to frequency. Also using unigrams with or without bigrams didn't make any significant improvements. The best accuracy achieved using decision trees was 68.1%. A comparison of accuracies obtained on the validation set using different features is shown in table 5.

3.4 Random Forest

I implemented random forest algorithm by using RandomForestClassifier from sklearn.ensemble provided by *scikit-learn*. I experimented using 10 estimators (trees) using both *presence* and *frequency* features. *presence* features performed better than *frequency* though the improvement was not substantial. A comparison of accuracies obtained on the validation set using different features is shown in table 5.

3.5 XGBoost

I also attempted tackling the problem with XGboost classifier. I set max tree depth to 25 where it refers to the maximum depth of a tree and is used to control over-fitting as a high value might result in the model learning relations that are tweetsd to the training data. Since XGboost is an algorithm that utilises an ensemble of Iaker trees, it is important to tune the number of estimators that is used. I realised that setting this value to 400 gave the best result. The best result was 0.78.72 which came from the configuration of presence with Unigrams + Bigrams.

3.6 SVM

I utilise the SVM classifier available in sklearn. I set the C term to be 0.1. C term is the penalty parameter of the error term. In other words, this influences the misclassification on the objective function. I run SVM with both Unigram as Ill Unigram + Bigram. I also run the configurations with frequency and presence. The best result was 81.55 which came the configuration of frequency and Unigram + Bigram.

3.7 Multi-Layer Perceptron

I used keras with TensorFlow backend to implement the Multi-Layer Perceptron model. I used a 1-hidden layer neural network with 500 hidden units. The output from the neural network

Algorithms	Presence		Frequency	
	Unigrams	Unigrams+Bigrams	Unigrams	Unigrams+Bigrams
Naive Bayes	78.16	79.68	77.52	79.38
Max Entropy	79.96	81.52	79.7	81.5
Decision Tree	68.1	68.01	67.82	67.78
Random Forest	76.54	77.21	76.16	77.14
XGBoost	77.56	78.72	77.42	78.32
SVM	79.54	81.11	79.83	81.55
MLP	80.1	81.7	80.15	81.35

Table 5: Comparison of various classifiers which use sparse vector representation

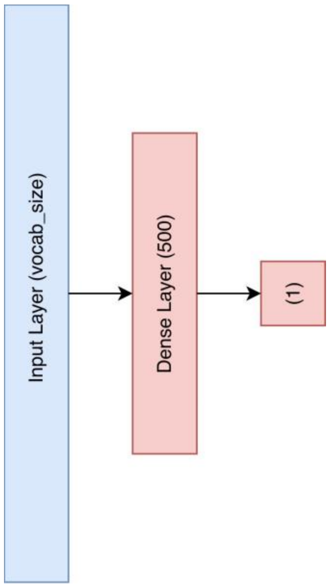

Figure 4: Architecture of the MLP Model.

is a single value which I pass through the sigmoid non-linearity to squish it in the range $[0,1]$. The sigmoid function is defined by $f(z) = \frac{1}{1+\exp^{-z}}$. The output from the neural network gives the probability $\Pr(positive|tweetst)$ i.e. the probability of the tweets sentiment being positive. At the prediction step, I round off the probability values to convert them to class labels 0 (negative) and 1 (positive). The architecture of the model is shown in figure . Red hidden layers represent layers with sigmoid non-linearity. I trained My model using binary cross entropy loss with the Iight update scheme being the one defined by Adam et. al. I also conducted experiments using SGD + Momentum Iight updates and found out that it takes too long to converge. I ran My model upto 20 epochs after which it began to overfit. I used sparse vector representation of tweets for training. I found that the presence of bigrams features significantly improved the accuracy.

- **Conclusion**

3.8 Summary of achievements

The provided tweets Ire a mixture of words, emoticons, URLs, hastags, user mentions, and symbols. Before training the I pre-process the tweets to make it suitable for feeding into models. I

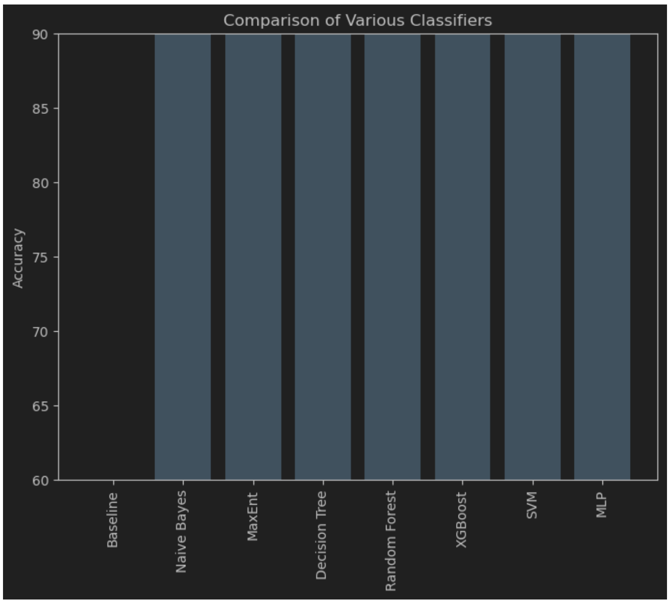

Figure 11: Comparison of accuracies of various models

implemented several machine learning algorithms like Naive Bayes, Maximum Entropy, Decision Tree, Random Forest, XGBoost, SVM, Multi-Layer Perceptron, Recurrent Neural networks. I used two types of features namely unigrams and bigrams for classification and observes that augmenting the feature vector with bigrams improved the accuracy. Once the feature has been extracted it was represented as either a sparse vector or a dense vector. It has been observed that *presence* in the sparse vector representation recorded a better performance than *frequency*.

Neural methods performed better than other classifiers in general.

Future directions

- *Handling emotion ranges*: I can improve and train My models to handle a range of sentiments. Tweets don't always have positive or negative sentiment. At times they may have no sentiment i.e. neutral. Sentiment can also have gradations like the sentence, *This is good*, is positive but the sentence, *This is extraordinary.* is somewhat more positive than the first. I can therefore classify the sentiment in ranges, say from -2 to +2.
- *Using symbols*: During My pre-processing, I discard most of the symbols like commas, full-stops, and exclamation mark. These symbols may be helpful in assigning sentiment to a sentence.