GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

TWITTER DATA SENTIMENTAL ANALYSIS USING MULTIPLE CLASSIFICATIONS

A Project Report of Capstone Project - 2

Submitted by

RAGHAV SHARMA

(1613107044 / 16SCSE107064)

in partial fulfillment for the award of the degree

of

Bachelor of Technology

IN

Computer Science and Engineering with Specialization of Business Analytics and Optimization

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of
Dr. A.JOHN
Assistant Professor

APRIL / MAY - 2020

GALGOTIAS
UNIVERSITY

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report **"TWITTER DATA SENTIMENTAL ANALYSIS USING MULTIPLE CLASSIFICATIONS"** is the bonafide work of **"RAGHAV SHARMA (1613107044)"** who carried out project work under my supervision.

SIGNATURE OF HEAD
Dr. MUNISH SHABARWAL,
PhD (Management), PhD (CS)
Professor & Dean
School of Computing Science and Engineering

SIGNATURE OF SUPERVISOR
Dr.A.JOHN

Assistant Professor
School of Computing Science and Engineering

TABLE OF CONTENT

ABSTRACT

Over the past decade humans have experienced exponential growth in the use of online resources, in particular social media and microblogging websites such as Twitter. This project focuses on implementing various machine learning and deep learning algorithms to perform sentimental analysis and comparing different machine learning methods for the task of sentiment classification. In the end, the major findings were that out of the different classification models evaluated it was found that the LSTM classifier provides the highest classification accuracy for this domain. From the evaluation of this study, it can be concluded that the proposed machine learning and deep learning techniques are effective for sentimental analysis.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 About the Project

What do you do when you want to express yourself or reach out to a large audience? We log on to one of our favourite social media services. Social Media has taken over in today's world, most of the methods we use to connect and communicate are using social networks, and Twitter is one of the major places where we express our sentiments about a specific topic or a concept. Twitter serves as a mean for individuals to express their thoughts or feelings about different subjects. These emotions are used in various analytics for better understanding of humans.

In this study, we will perform a sentimental analysis of "tweets" using diverse machine learning algorithms. In this, we try to classify the polarity of the tweet where it is either positive or negative. If the tweet has both positive and negative features, the more superior sentiment should be preferred as the final label.

1.2 Machine Learning

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.

In general, there are two types of machine learning algorithms supervised, and unsupervised, there are variations on these algorithms such hybrid types (semi-supervised learning) but for this report they will be classified into one of the two categories.

The method of supervised learning consists of presenting an algorithm with a training dataset; this dataset consists of training examples and the corresponding expected output for each example. The expected output is general known as the target. A supervised learning algorithm uses this dataset so that it can learn to map the input examples to their expected target. If the training process is implemented correctly the machine learning algorithm should be able to generalize the training data so that it can correctly map new data that it has never seen before.

Unsupervised machine learning algorithms do not require training data, they operate on data where the output is unknown. The object of this form of learning is usually to discover patterns in the data that may not be known by the researcher. An example of an unsupervised method would be clustering where the algorithm uses a distance function to group similar data points together.

This project focuses exclusively on supervised learning algorithms for the task of text classification, however there are two major applications for supervised methods:

Classification:

The target output is a class or label, the simplest case is a choice between zero ore one, although there can also be multiple alternative classes. Classification is used for many applications such as test classification, object recognition, and voice recognition software.

Regression:

In this case the target is a real number or vector of real numbers. Regression is mostly used for prediction. Supervised regression algorithms are used mostly for prediction. Example applications are stock market prediction, in power systems analysis it can be to predict spikes in a network, and most recently they have been used in memory caching to complement the ‘locality of reference’ method.

1.3 Sentimental Analysis

Sentiment analysis is predominantly implemented in software which can autonomously extract emotions and opinions in text. It has many real-world applications such as it allows companies to analyse how their products or brand is being perceived by their consumers, this usage is particularly applicable to this project. It is difficult to classify sentiment analysis as one specific field of study as in incorporates many different areas such as linguistics, Natural Language Processing (NLP), and Machine Learning or Artificial Intelligence.

As the majority of the sentiment that is uploaded to the internet is of an unstructured nature it is a difficult task for computers to process it and extract meaningful information from it. Natural language processing techniques are used to transform this raw data into a form that it can be processed efficiently by a computer.

CHAPTER 2

EXISTING SYSTEM

Over the past decade humans have experienced exponential growth in the use of online resources, in particular social media and microblogging websites such as Twitter, Facebook and YouTube. Many companies and organizations have identified these resources as a rich mine of marketing knowledge. Traditionally companies used interviews, questionnaires and surveys to gain feedback and insight into how customers felt about their products. These traditional methods were often extremely time consuming and expensive and did not always return the results that the companies were looking for due to environmental factors and poorly designed surveys.

Natural language processing and sentiment analysis are playing an increasingly important role in making educated decisions on marketing strategies and giving valuable feedback on products and services. There are massive amounts of data containing consumer sentiment uploaded to the internet every day, this type of data is predominantly unstructured text that is difficult for computers to gain meaning from.

In the past it was not possible to process such large amounts of unstructured data but now with computational, massive datasets can be now processed with relative ease.

CHAPTER 3

PROPOSED SYSTEM

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 parties 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, we are able improve the accuracy of our sentiment analysis predictions.

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

we use the datasets from Kaggle which was crawled from the internet and labeled positive/negative. The data provided comes with emoticons (emoji), usernames and hashtags which are required to be processed (so as to be readable) and converted into a standard form. We also need to extract useful features from the text such unigrams and bigrams which is a form of representation of the “tweet”.

We use various machine learning algorithms based on NLP (Natural Language Processing) to conduct sentiment analysis using the extracted features. Finally, we report our experimental results and findings at the end.

CHAPTER 4

IMPLEMENTATION

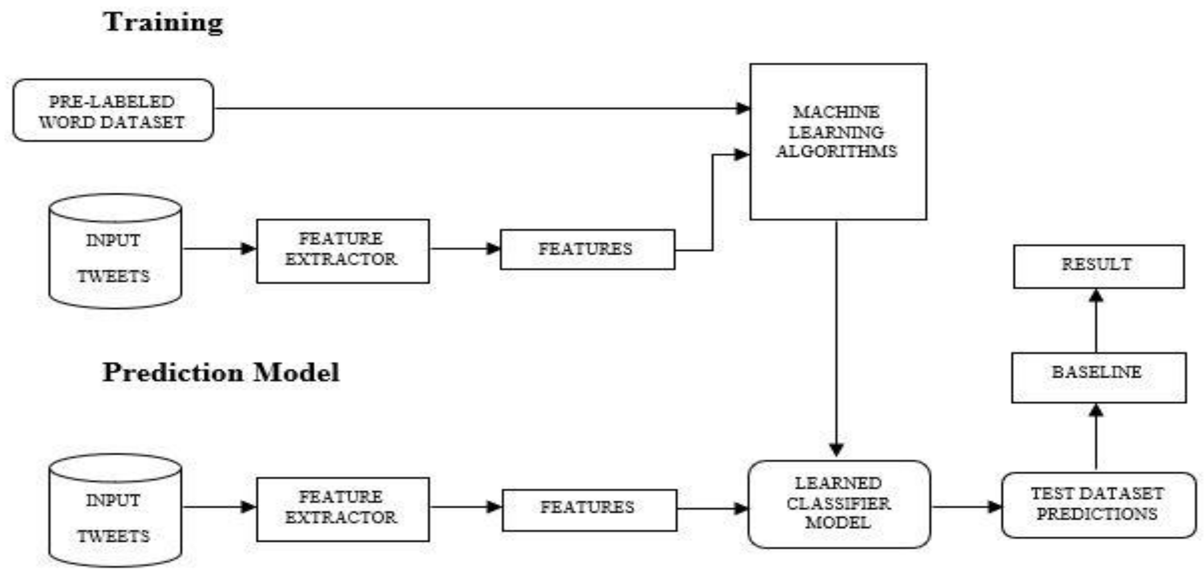


Figure 1. Machine Learning Model

4.1 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. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. We first do some general pre-processing on tweets which is as follows.

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

	Total	Unique	Average	Max	Positive	Negative
Tweets	100000	—	—	—	56462	43538
User Mention	88955	—	0.8895	12	—	—
Emoticons	1184	—	0.0118	5	997	187
URLs	3599	—	0.0360	4	—	—
Unigrams	1192636	50376	11.9264	40	—	—
Bigrams	1093168	385113	10.9317	—	—	—

Table 1. Statistics of Preprocessed Train dataset

	Total	Unique	Average	Max	Positive	Negative
Tweets	300000	—	—	—	—	—
User Mention	297470	—	0.9916	12	—	—
Emoticons	3410	—	0.0114	5	3043	367
URLs	9486	—	0.0316	4	—	—
Unigrams	3626166	93456	12.0872	40	—	—
Bigrams	3327647	879825	11.0922	—	—	—

Table 2. Statistics of Preprocessed Test dataset

We handle special twitter features as follows.

4.1.1 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, we replace all the URLs in tweets with the word URL. The regular expression used to match URLs is ((www\.[\S]+)|(https?://[\S]+)).

4.1.2 User Mention

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

4.1.3 Emoticon

Users often use a number of different emoticons in their tweet 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, we match some common emoticons which are used very frequently. We 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 our method is given in table.

EMOTICONS	TYPE	REGEX	REPLACEMENT
:) , :) , :-) , (: , (: , (- : , : ‘)	Smile	(: \s? \) |: -\) |\(\s? :|\(-:|:\’\))	EMO_POS
:D, : D , :-D , xD, x-D , XD , X-D	Laugh	(: \s? D|:-D|x-? D|X-? D)	EMO_POS
;-) , ;) , ;-D , ;D , (; , (-;	Wink	(: \s? \ (|: -\ (|\) \s? :|\)-:)	EMO_POS
<3 , :*	Love	(<3|: *)	EMO_POS
:-(, : (, : (,): ,)-:	Sad	(: \s? \ (|: -\ (|\) \s? :|\)-:)	EMO_NEG
:,(, :’(, :"(	Cry	(:,\(|:\’\(|:"\()	EMO_NEG

Table 3. Emoticons matched by our method

4.1.4 Hashtags

Hashtags are unspaced phrases prefixed by the hash symbol (#) which is frequently used by users to mention a trending topic on twitter. We 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+).

4.1.5 Re-tweet

Retweets are tweets which have already been sent by someone else and are shared by other users. Retweets begin with the letters RT. We 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 tweet level pre-processing, we 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 so 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. We 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.

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 you are so chill out person :)
Normalized	USER_MENTION hii you are so chill out person EMO_POS
Raw	@bolly47 oh no :’(r.i.p. your bella
Normalized	USER_MENTION oh no EMO_NEG r.i.p your bella

Table 4. Tweets in Raw and Normalized version

4.2 Feature Extraction

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

4.2.1 Unigram

Probably the simplest and the most commonly used features for text classification is the presence of single words or tokens in the text. We extract single words from the training dataset and create a frequency distribution of these words. A total of 50376 unique words are extracted from 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. We, therefore, only use top N words from these to create our vocabulary where N is 15000 for sparse vector classification and 90000 for dense vector classification. The frequency distribution of top 20 words in our vocabulary is shown in figure 1.

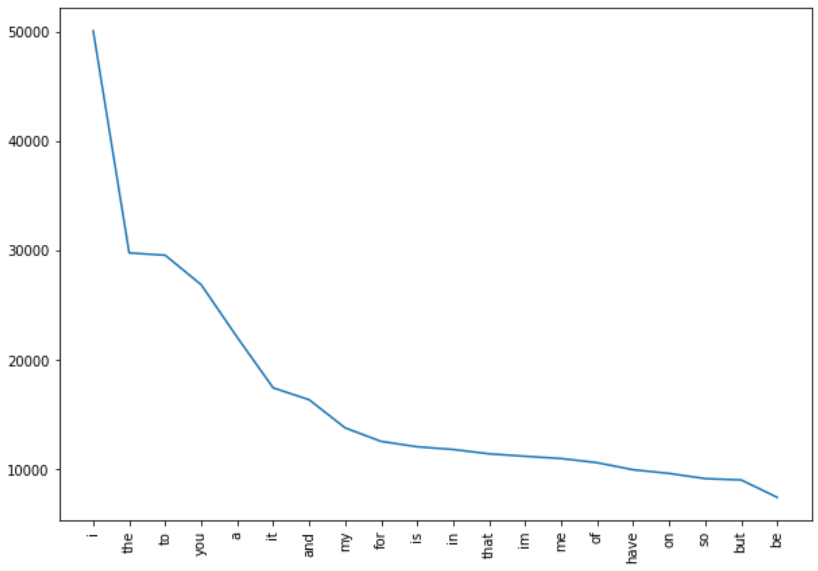


Figure 1. Frequencies of Top 20 Unigrams

We 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 $\log(\text{Frequency}) = -0.78\log(\text{Rank}) + 13.31$.

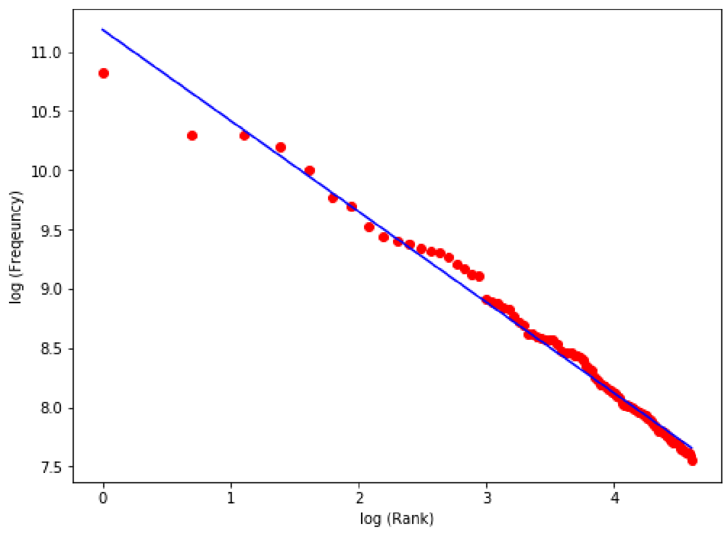


Figure 2. Unigrams frequencies follow Zipf's Law

4.2.2 Bigram

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 385113 unique bigrams were 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. We therefore use only top 10000 bigrams from these to create our vocabulary. The frequency distribution of top 20 bigrams in our vocabulary is shown in figure 3.

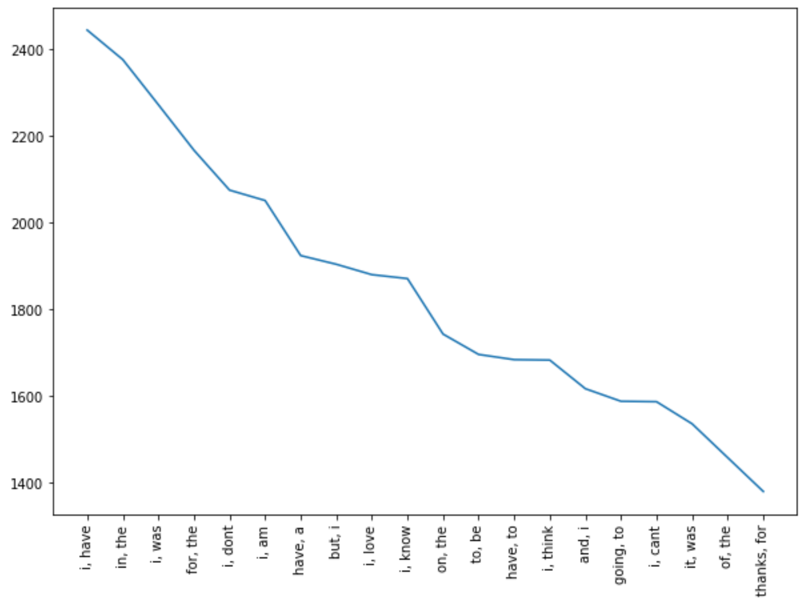


Figure 3. Frequencies of Top 20 Bigrams

4.3 Feature Representation

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

4.3.1 Sparse Vector Representation

Depending on whether or not we are using bigram features, the sparse vector representation of each tweet 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 tweet has a positive value at the indices of unigrams (and bigrams) which are present in that tweet 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 we 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 tweet 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 tweet 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.

4.3.2 Dense Vector Representation

For dense vector representation we use a vocabulary of unigrams of size 90000 i.e. the top 90000 words in the dataset. We 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 tweet is then represented by a vector of these indices which is a dense vector.

4.4 Classifiers

4.4.1 Naïve Bayes

Naive Bayes is a simple model that can be used for text classification. In this model, the class ĉ is assigned to a tweet *t*, where

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

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

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.

4.4.2 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, we can add features like bigrams without worrying about feature overlap. In a binary classification problem like the one we 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 tweet, and λ is the weight vector. The weight vector is found by numerical optimization of the lambdas to maximize the conditional probability.

4.4.3 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 represent the final classes of the data points. It is a supervised classifier model that uses data with known labels to form the decision tree and then the model is applied to the test data. For each node in the tree, the best test condition or decision has to be taken. We 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. We choose a split that minimizes the GINI factor.

4.4.4 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,...x_n and their respective sentiment labels y_1,y_2,...y_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...B. Finally, a majority vote is taken of predictions of these B trees.

4.4.5 XGBoost

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

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

where F is the space of trees, x_i is the input and $\hat{y}_i$ is the final output. We 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.

4.4.6 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* are the class, we 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 follows

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

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

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

to separate the points well.

4.4.7 Multi-Layer Perceptron

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

4.4.8 Convolutional Neural Network

Convolutional Neural Networks or CNNs are a type of neural network which involves layers called convolution layers that can interpret spacial data. A convolution layer has several filters or kernels which it learns to extract specific types of features from the data. The kernel is a 2D window which is slid over the input data performing the convolution operation. We use temporal convolution in our experiments which is suitable for analyzing sequential data like tweets.

4.4.9 Recurrent Neural Network

Recurrent Neural Network is 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 a memory of the network and learns contextual information which is important for the 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 the information. We used Long Term Short Memory (LSTM) networks in our experiments which are a special kind of RNN capable of remembering information over a long time.

4.5 Experiment

We perform experiments using various different classifiers. Unless otherwise specified, we use 10% of the training dataset for validation of our models to check against overfitting i.e. we use 100000 tweets for training and 80000 tweets for validation. For Naive Bayes, Maximum Entropy, Decision Tree, Random Forest, XGBoost, SVM and Multi-Layer Perceptron we use sparse vector representation of tweets. For Recurrent Neural Networks and Convolutional Neural Network, we use the dense vector representation.

4.5.1 Baseline

For a baseline, we use a simple positive and negative word counting method to assign sentiment to a given tweet. We use the Opinion Dataset of positive and negative words to classify tweets. In cases when the number of positive and negative words are equal, we assign positive sentiment. Using this baseline model, we achieve a classification accuracy of 65.35%.

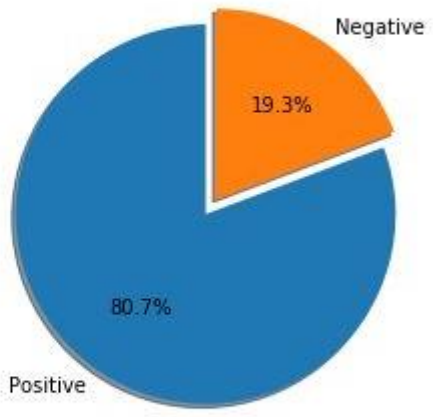


Figure 4. Prediction of Baseline Model

4.5.2 Naïve Bayes

We used MultinomialNB from sklearn.naive_bayes package of scikit-learn for Naive Bayes classification. We used Laplace smoothed version of Naive Bayes with the smoothing parameter α set to its default value of 1. We used sparse vector representation for classification and ran experiments using both presence and frequency feature types. We found that presence features outperform frequency features because Naive Bayes is essentially built to work better on integer features rather than floats. We also observed that addition of bigram features improves the accuracy. We obtain a best validation accuracy of 77.34% using Naive Bayes with presence of unigrams and bigrams.

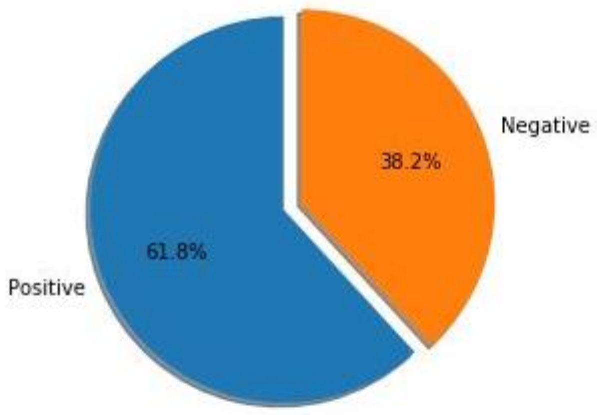


Figure 5. Prediction of Naïve Bayes Model

4.5.3 Maximum Entropy

The nltk library provides several text analysis tools. We use the MaxentClassifier to perform sentiment analysis on the given tweets. Unigrams, bigrams and a combination of both were 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 76.27% compared to just unigrams (75.64%) and just bigrams (75.40%).

For a binary classification problem, Logistic Regression is essentially the same as Maximum Entropy. So, we 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 we get almost the same accuracies, but the performance is slightly better when we use unigrams and bigrams together. The best accuracy achieved was 77.87%.

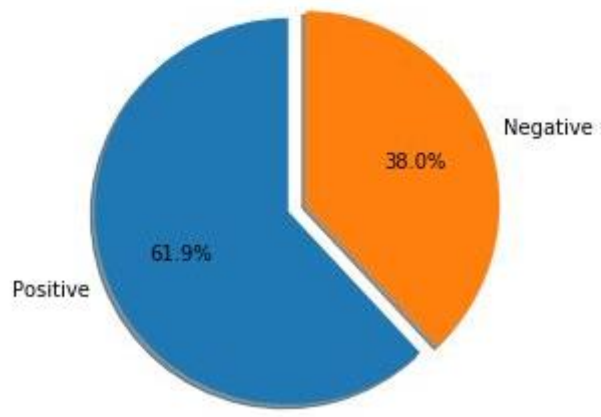


Figure 6. Prediction of Maxent Model

4.5.4 Decision Tree

We use the DecisionTreeClassifier from sklearn.tree package provided by scikit-learn to build our 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 69.06%.

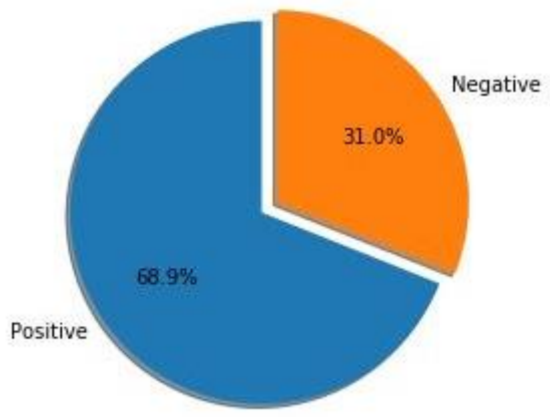


Figure 7. Prediction of Decision Tree Model

4.5.5 Random Forest

We implemented random forest algorithm by using RandomForestClassifier from sklearn.ensemble provided by scikit-learn. We experimented using 10 estimators (trees) using both presence and frequency features. presence features performed better than frequency though the improvement was not substantial. We get an accuracy of 76.23%.

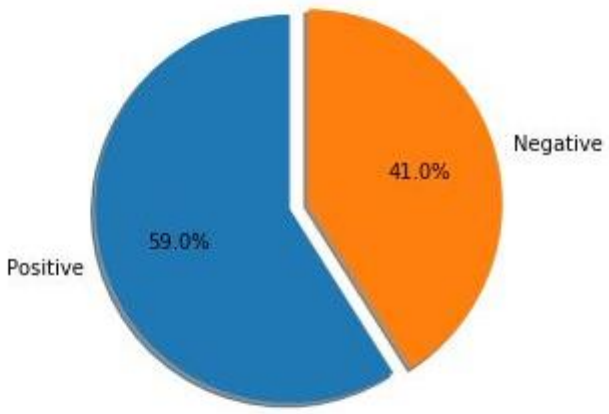


Figure 8. Prediction of Random Forest Model

4.5.6 XGBoost

We also attempted tackling the problem with XGboost classifier. We 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 tied to the training data. Since XGboost is an algorithm that utilises an ensemble of weaker trees, it is important to tune the number of estimators that is used. We realised that setting this value to 400 gave the best result. The best result was 77.11% which came from the configuration of presence with Unigrams + Bigrams.

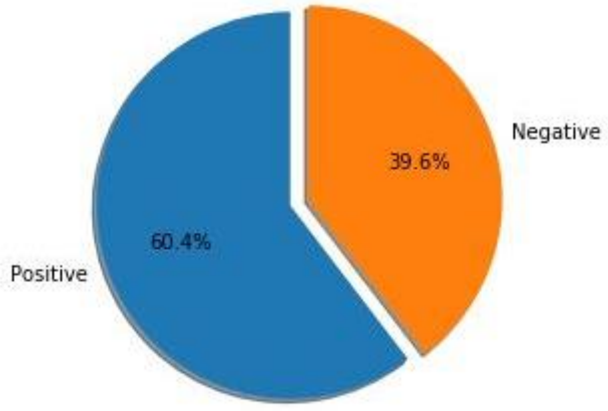


Figure 9. Prediction of XGBoost Model

4.5.7 SVM

We utilise the SVM classifier available in sklearn. We 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. We run SVM with both Unigram as well Unigram + Bigram. We also run the configurations with frequency and presence. The best result was 78.73% which came with the configuration of frequency and Unigram + Bigram.

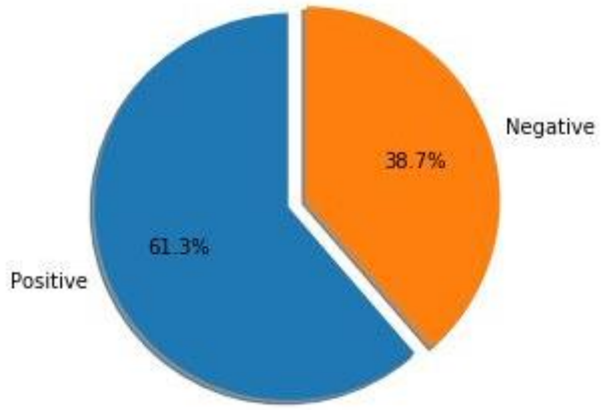


Figure 10. Prediction of SVM Model

4.5.8 Multi-Layer Perceptron

We used keras with TensorFlow backend to implement the Multi-Layer Perceptron model. We used a 1-hidden layer neural network with 500 hidden units. The output from the neural network is a single value which we pass through the sigmoid non-linearity to squish it in the range [0,1]. The architecture of the model is shown in figure. Red hidden layers represent layers with sigmoid non-linearity. We trained our model using binary cross entropy loss with the weight update scheme being the one defined by Adam et. al. We ran our model up to 20 epochs after which it began to overfit. We used sparse vector representation of tweets for training. We found that the presence of bigrams features significantly improved the accuracy. We achieved the best accuracy of 76.48%.

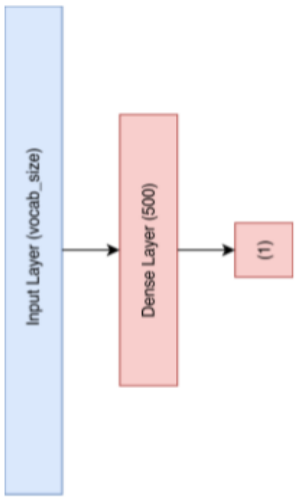


Figure 11. Architecture of MLP model

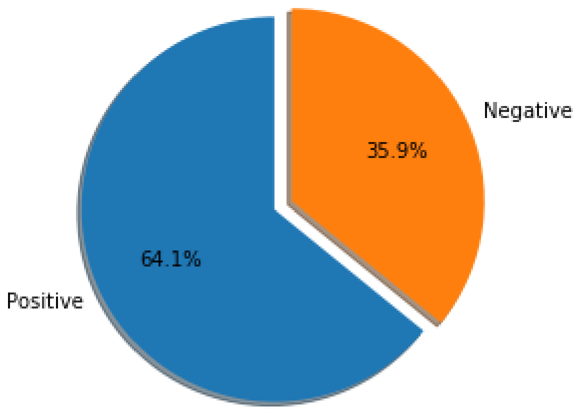


Figure 12. Prediction of MLP Model

4.5.9 Convolutional Neural Network

We used keras with TensorFlow backend to implement the Convolutional Neural Network model. We used the dense vector representation of the tweets to train our CNN models. We used a vocabulary of top 90000 words from the training dataset. We represent each word in our vocabulary with an integer index from 1...90000 where the integer index represents the rank of the word in the dataset. The integer index 0 is reserved for the special padding word. Further each of these 90000+1 words is represented by a 200-dimensional vector. The first layer of our models is the Embedding layer which is a matrix of shape (v+1)×d where v is vocabulary size (=90000) and d is the dimension of each word vector (=200). We initialize the embedding layer with random weights from N(0, 0.01). Each row of this embedding matrix represents the 200-dimensional word vector for a word in the vocabulary. For words in our vocabulary which match Glove word vectors provided by the StanfordNLP group, we seed the corresponding row of the embedding matrix from Glove vectors. Each tweet i.e. its dense vector representation is padded with 0s at the end until its length is equal to max_length which is a parameter we tweak in our experiments. We trained our model using binary cross entropy loss with the weight update scheme being the one defined by Adam et. al. We also conducted experiments using SGD + Momentum weight updates and found out that it takes longer (≈100 epochs) to converge compared to validation accuracy equivalent to Adam. We ran our model up to 10 epochs. Using the Adam weight update scheme, the model converges very fast (≈4 epochs) and begins to overfit badly after that. We, therefore, use models from 3rd or 4th epoch for our results. We tried four different CNN architectures which are as follows.

1-Conv-NN: The complete architecture of the network is embedding_layer (800001×200) → dropout (0.2) → conv_1 (500 filters) → relu → global_maxpool → dense (500) → relu → dropout (0.2) → dense (1) → sigmoid

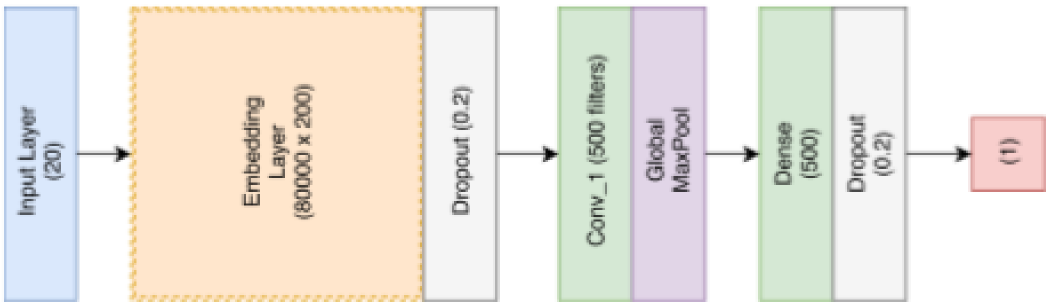


Figure 13. Neural Network Architecture with 1 Conv Layer

2-Conv-NN: The complete architecture of the network is embedding_layer (900001×200) → dropout (0.4) → conv_1 (600 filters) → relu → conv_2 (300 filters) → relu → flatten → dense (600) → relu → dropout (0.5) → dense (1) → sigmoid

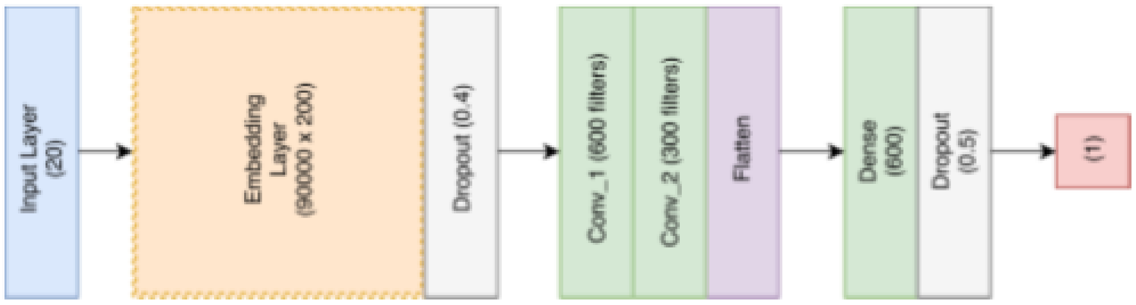


Figure 14. Neural Network Architecture with 2 Conv Layers

3-Conv-NN: The complete architecture of the network is embedding_layer (900001×200) → dropout (0.4) → conv_1 (600 filters) → relu → conv_2 (300 filters) → relu → conv_3 (150 filters) → relu → flatten → dense (600) → relu → dropout (0.5) → dense (1) → sigmoid

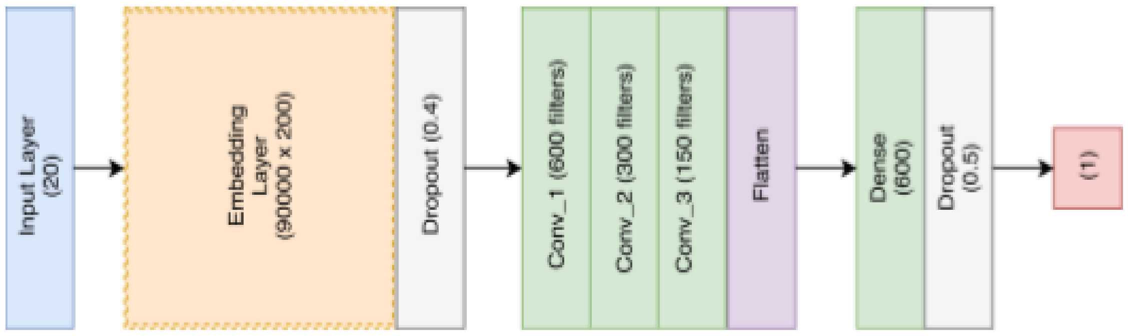


Figure 15. Neural Network Architecture with 3 Conv Layers

4-Conv-NN: The complete architecture of the network is embedding_layer (900001×200) → dropout (0.4) → conv_1 (600 filters) → relu → conv_2 (300 filters) → relu → conv_3 (150 filters) → relu → conv_4 (75 filters) → relu → flatten → dense (600) → relu → dropout (0.5) → dense (1) → sigmoid

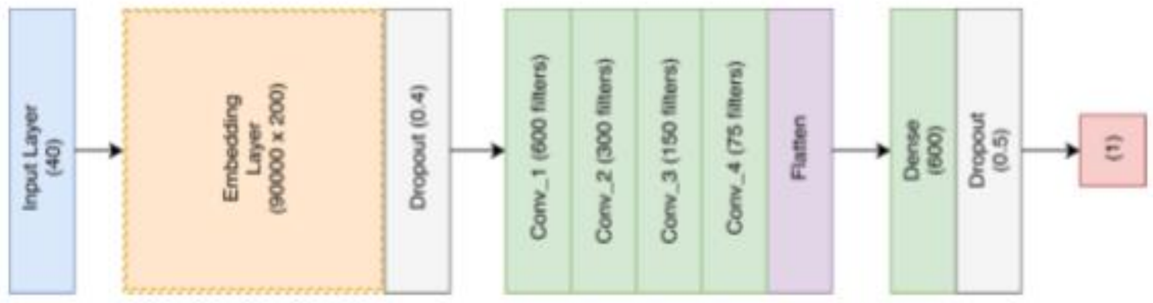


Figure 16. Neural Network Architecture with 4 Conv Layers

We notice that each successive CNN model is better than the previous one with 1-Conv-NN, 2-Conv-NN, 3-Conv-NN and 4-Conv-NN achieving accuracies of 75.86, 76.76, 76.95 and 78.47 respectively.

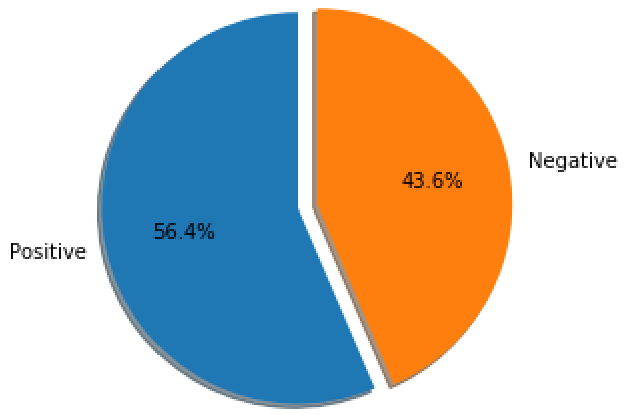


Figure 17. Prediction of best CNN Model

4.5.10 Recurrent Neural Networks

We used neural networks with LSTM layers in our experiments. We used a vocabulary of top 20000 words from the training dataset. We used the dense vector representation for training our models. We pad or truncate each dense vector representation to make it equal to max_length which is a parameter we tweak in our experiments. We obtain best accuracy of 78.87% among the different LSTM models.

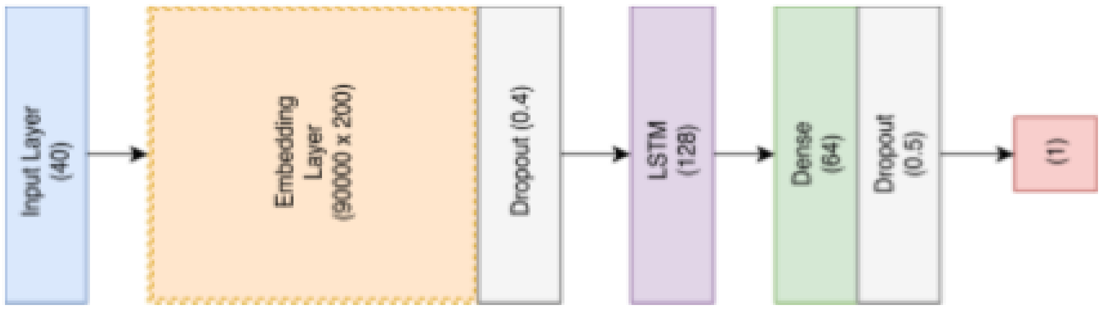


Figure 18. Architecture of best performing LSTM – NN

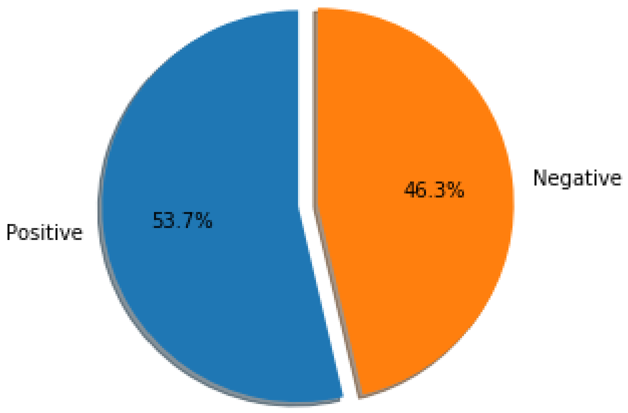
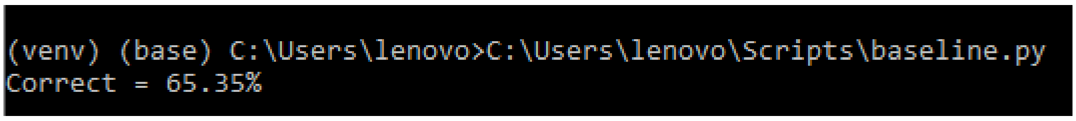


Figure 19. Prediction of best LSTM Model

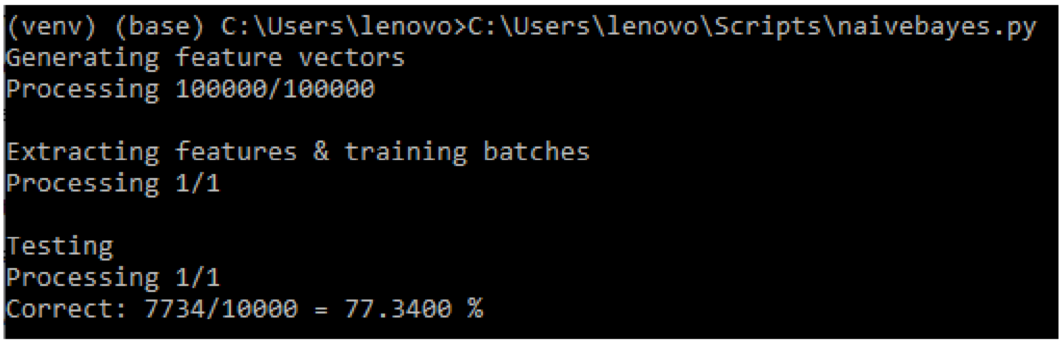
CHAPTER 5

RESULT



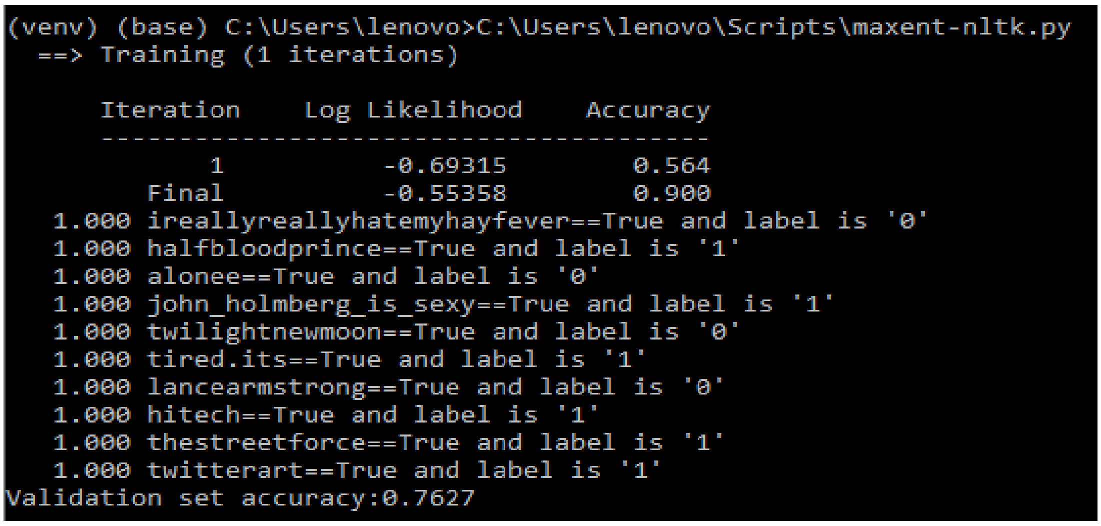
```
(venv) (base) C:\Users\lenovo>C:\Users\lenovo\Scripts\baseline.py
Correct = 65.35%
```

Figure 20. Accuracy achieved by Baseline Model



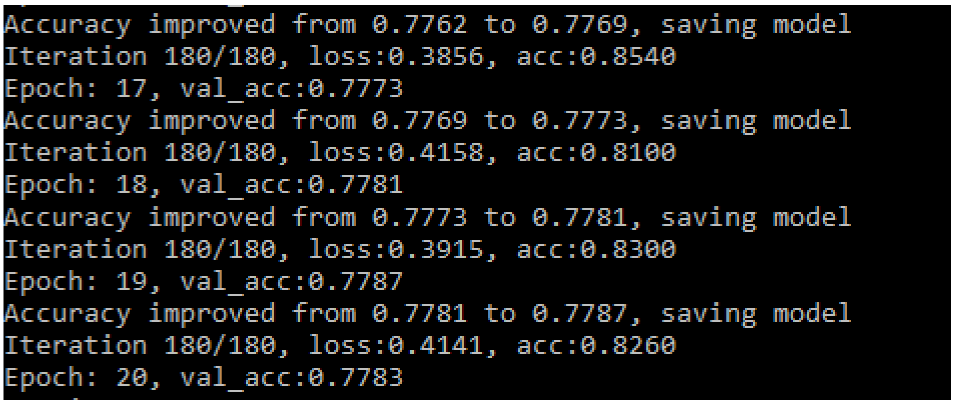
```
(venv) (base) C:\Users\lenovo>C:\Users\lenovo\Scripts\naivebayes.py
Generating feature vectors
Processing 100000/100000

Extracting features & training batches
Processing 1/1

Testing
Processing 1/1
Correct: 7734/10000 = 77.3400 %
```

Figure 21. Accuracy achieved by Naïve Bayes Model



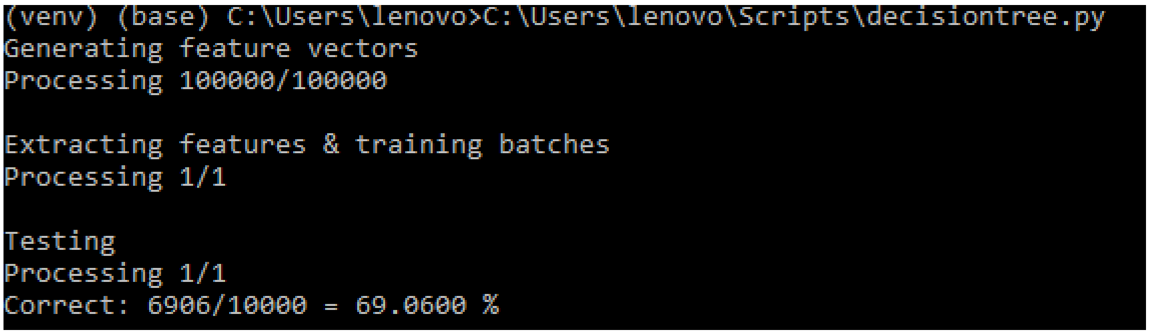
```
(venv) (base) C:\Users\lenovo>C:\Users\lenovo\Scripts\maxent-nltk.py
  ==> Training (1 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
             1          -0.69315        0.564
         Final          -0.55358        0.900
   1.000 ireallyreallyhatemyhayfever==True and label is '0'
   1.000 halfbloodprince==True and label is '1'
   1.000 alonee==True and label is '0'
   1.000 john_holmberg_is_sexy==True and label is '1'
   1.000 twilightnewmoon==True and label is '0'
   1.000 tired.its==True and label is '1'
   1.000 lancearmstrong==True and label is '0'
   1.000 hitech==True and label is '1'
   1.000 thestreetforce==True and label is '1'
   1.000 twitterart==True and label is '1'
Validation set accuracy:0.7627
```

Figure 22. Accuracy achieved by Maxent using Unigrams + Bigrams



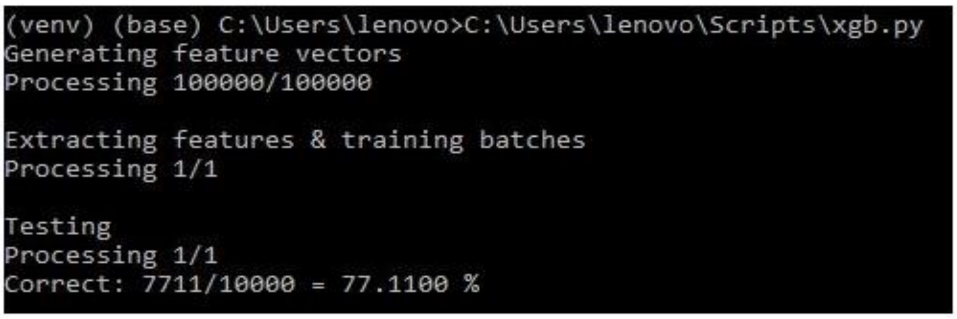
```
Accuracy improved from 0.7762 to 0.7769, saving model
Iteration 180/180, loss:0.3856, acc:0.8540
Epoch: 17, val_acc:0.7773
Accuracy improved from 0.7769 to 0.7773, saving model
Iteration 180/180, loss:0.4158, acc:0.8100
Epoch: 18, val_acc:0.7781
Accuracy improved from 0.7773 to 0.7781, saving model
Iteration 180/180, loss:0.3915, acc:0.8300
Epoch: 19, val_acc:0.7787
Accuracy improved from 0.7781 to 0.7787, saving model
Iteration 180/180, loss:0.4141, acc:0.8260
Epoch: 20, val_acc:0.7783
```

Figure 23. Accuracy achieved by Logistic Regression Model



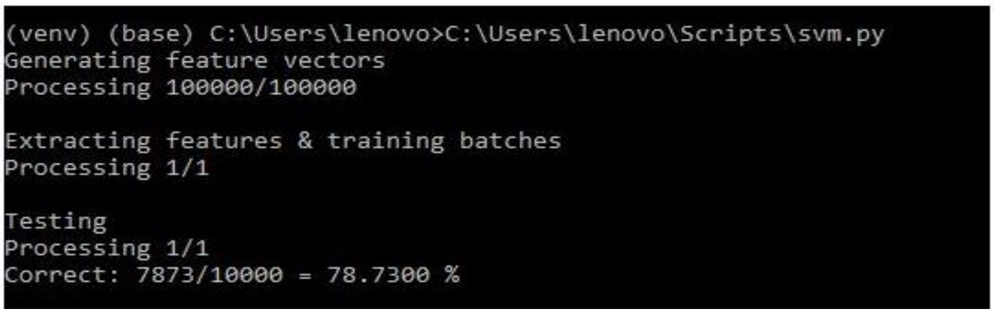
```
(venv) (base) C:\Users\lenovo>C:\Users\lenovo\Scripts\decisiontree.py
Generating feature vectors
Processing 100000/100000

Extracting features & training batches
Processing 1/1

Testing
Processing 1/1
Correct: 6906/10000 = 69.0600 %
```

Figure 24. Accuracy achieved by Decision Tree Model



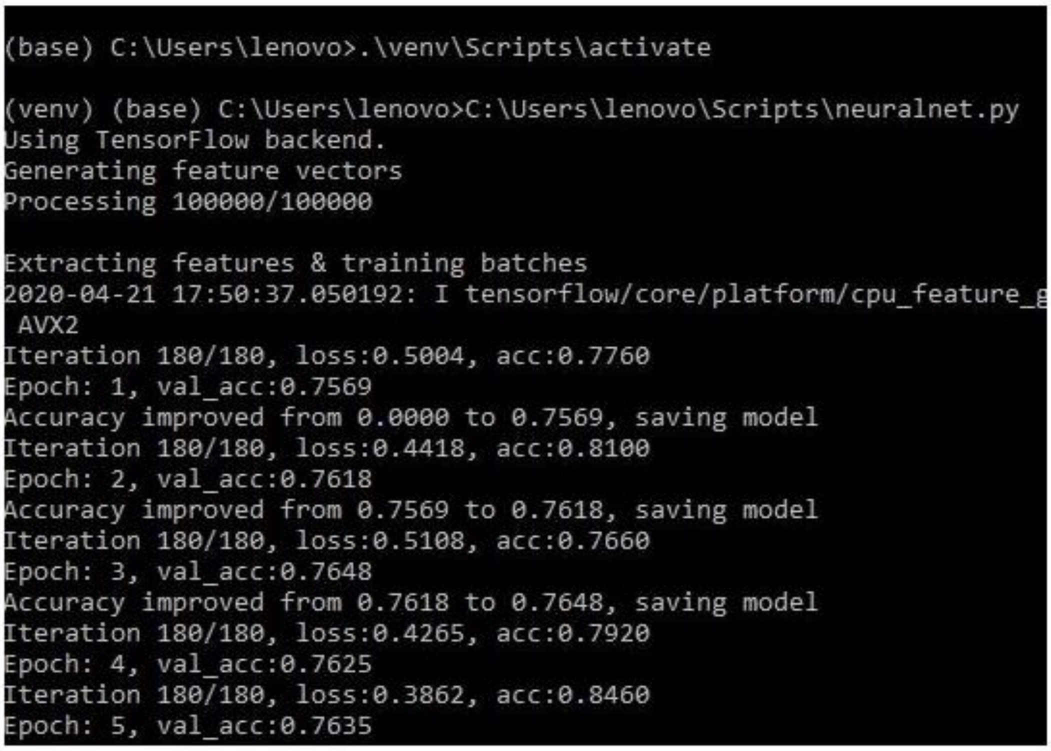
```
(venv) (base) C:\Users\lenovo>C:\Users\lenovo\Scripts\xgb.py
Generating feature vectors
Processing 100000/100000

Extracting features & training batches
Processing 1/1

Testing
Processing 1/1
Correct: 7711/10000 = 77.1100 %
```

Figure 25. Accuracy achieved by XGBoost Model



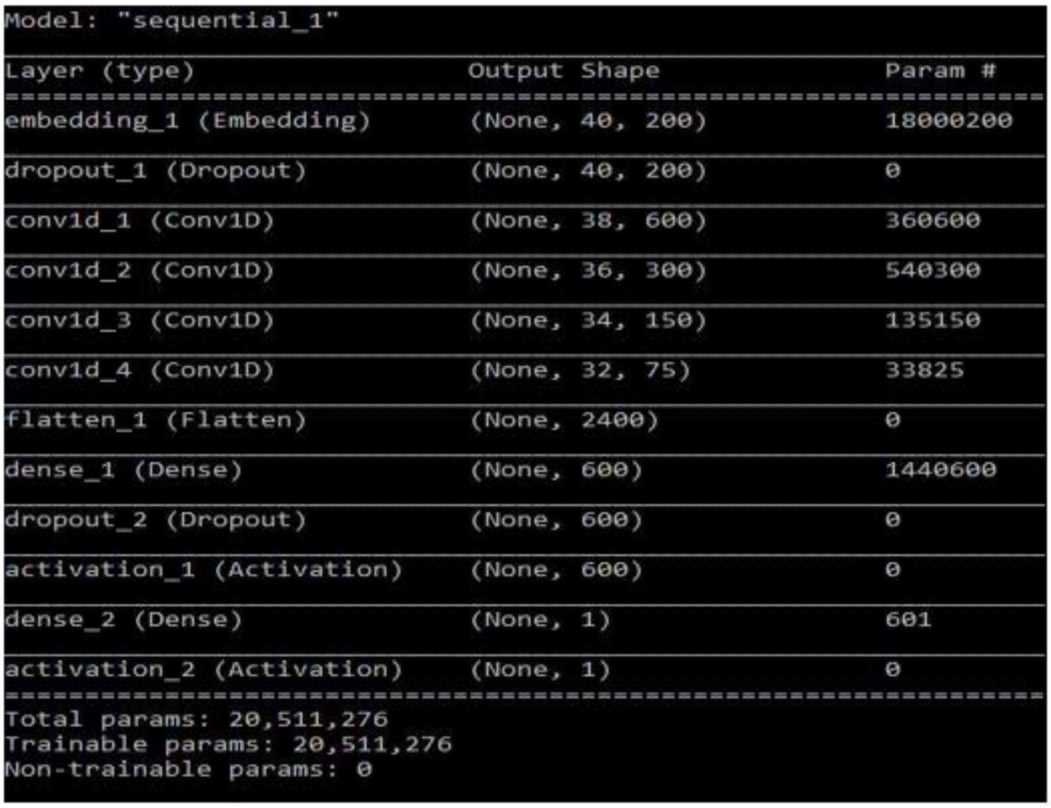
```
(venv) (base) C:\Users\lenovo>C:\Users\lenovo\Scripts\svm.py
Generating feature vectors
Processing 100000/100000

Extracting features & training batches
Processing 1/1

Testing
Processing 1/1
Correct: 7873/10000 = 78.7300 %
```

Figure 26. Accuracy achieved by SVM Model



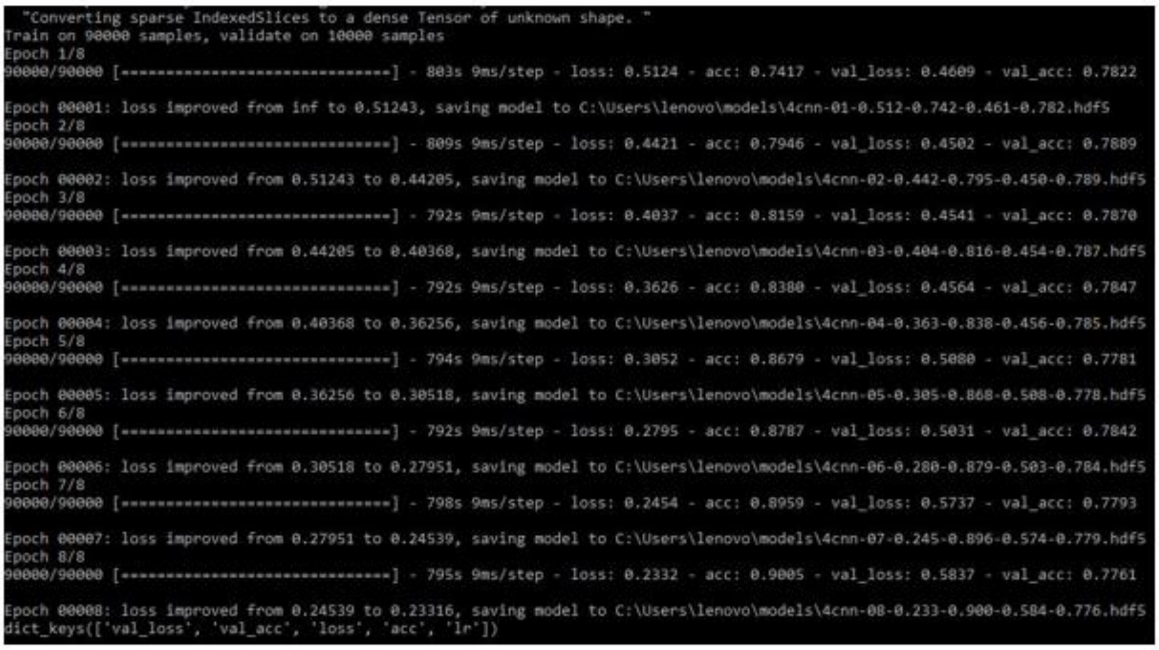
```
(base) C:\Users\lenovo>.\venv\Scripts\activate

(venv) (base) C:\Users\lenovo>C:\Users\lenovo\Scripts\neuralnet.py
Using TensorFlow backend.
Generating feature vectors
Processing 100000/100000

Extracting features & training batches
2020-04-21 17:50:37.050192: I tensorflow/core/platform/cpu_feature_g
 AVX2
Iteration 180/180, loss:0.5004, acc:0.7760
Epoch: 1, val_acc:0.7569
Accuracy improved from 0.0000 to 0.7569, saving model
Iteration 180/180, loss:0.4418, acc:0.8100
Epoch: 2, val_acc:0.7618
Accuracy improved from 0.7569 to 0.7618, saving model
Iteration 180/180, loss:0.5108, acc:0.7660
Epoch: 3, val_acc:0.7648
Accuracy improved from 0.7618 to 0.7648, saving model
Iteration 180/180, loss:0.4265, acc:0.7920
Epoch: 4, val_acc:0.7625
Iteration 180/180, loss:0.3862, acc:0.8460
Epoch: 5, val_acc:0.7635
```

Figure 27. Accuracy achieved by MLP Model



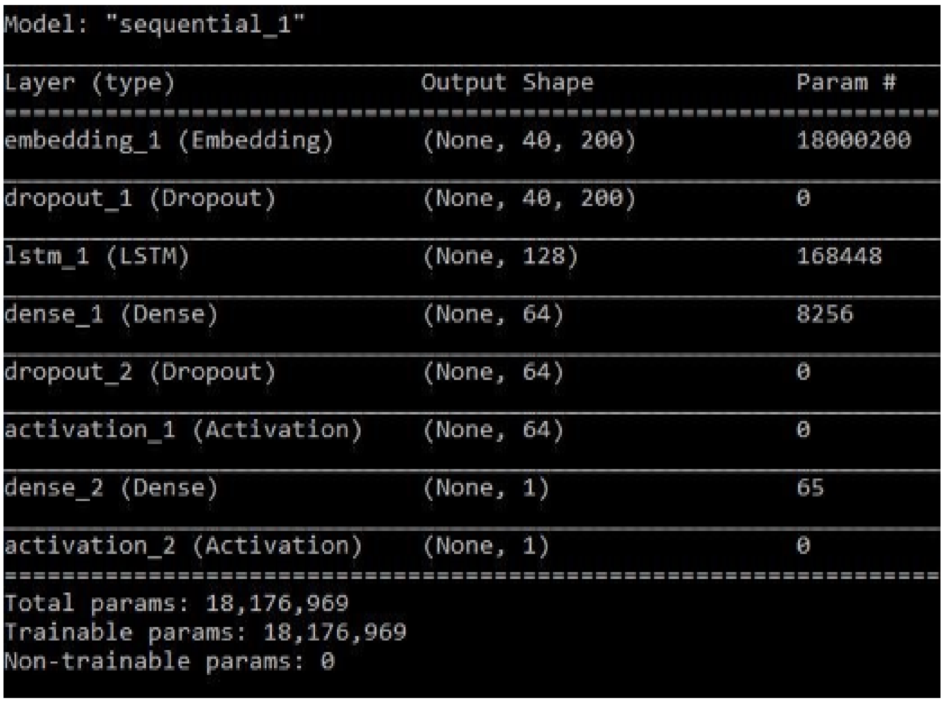
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 40, 200)           18000200
_________________________________________________________________
dropout_1 (Dropout)          (None, 40, 200)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 38, 600)           360600
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 36, 300)           540300
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 34, 150)           135150
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 32, 75)            33825
_________________________________________________________________
flatten_1 (Flatten)          (None, 2400)              0
_________________________________________________________________
dense_1 (Dense)              (None, 600)               1440600
_________________________________________________________________
dropout_2 (Dropout)          (None, 600)               0
_________________________________________________________________
activation_1 (Activation)    (None, 600)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 601
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 20,511,276
Trainable params: 20,511,276
Non-trainable params: 0
```

Figure 28. Params in 4 Conv-NN



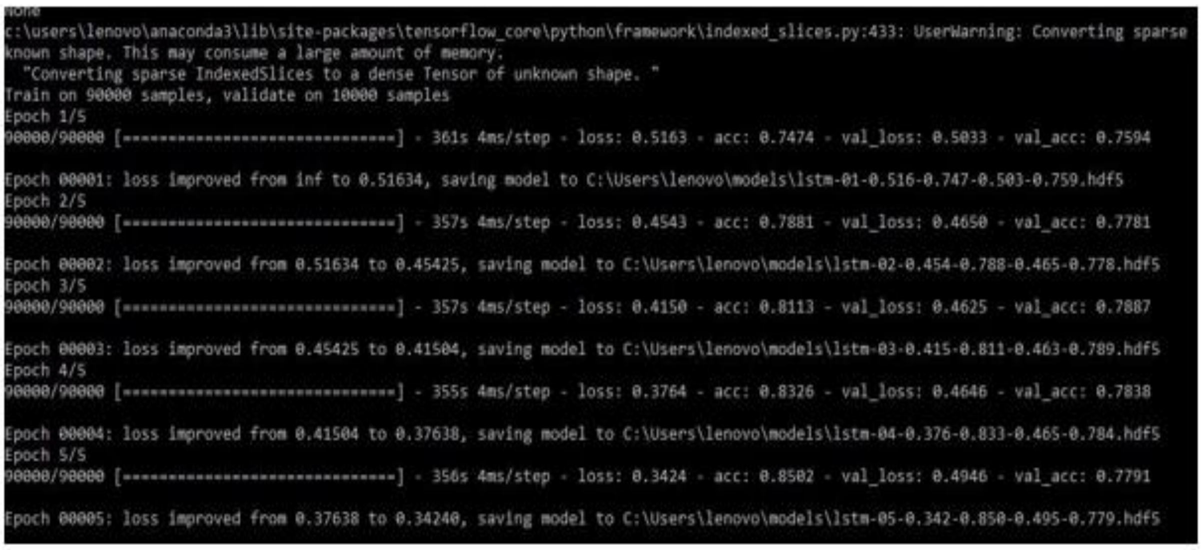
```
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 90000 samples, validate on 10000 samples
Epoch 1/8
90000/90000 [==============================] - 803s 9ms/step - loss: 0.5124 - acc: 0.7417 - val_loss: 0.4609 - val_acc: 0.7822

Epoch 00001: loss improved from inf to 0.51243, saving model to C:\Users\lenovo\models\4cnn-01-0.512-0.742-0.461-0.782.hdf5
Epoch 2/8
90000/90000 [==============================] - 809s 9ms/step - loss: 0.4421 - acc: 0.7946 - val_loss: 0.4502 - val_acc: 0.7889

Epoch 00002: loss improved from 0.51243 to 0.44205, saving model to C:\Users\lenovo\models\4cnn-02-0.442-0.795-0.450-0.789.hdf5
Epoch 3/8
90000/90000 [==============================] - 792s 9ms/step - loss: 0.4037 - acc: 0.8159 - val_loss: 0.4541 - val_acc: 0.7870

Epoch 00003: loss improved from 0.44205 to 0.40368, saving model to C:\Users\lenovo\models\4cnn-03-0.404-0.816-0.454-0.787.hdf5
Epoch 4/8
90000/90000 [==============================] - 792s 9ms/step - loss: 0.3626 - acc: 0.8380 - val_loss: 0.4564 - val_acc: 0.7847

Epoch 00004: loss improved from 0.40368 to 0.36256, saving model to C:\Users\lenovo\models\4cnn-04-0.363-0.838-0.456-0.785.hdf5
Epoch 5/8
90000/90000 [==============================] - 794s 9ms/step - loss: 0.3052 - acc: 0.8679 - val_loss: 0.5080 - val_acc: 0.7781

Epoch 00005: loss improved from 0.36256 to 0.30518, saving model to C:\Users\lenovo\models\4cnn-05-0.305-0.868-0.508-0.778.hdf5
Epoch 6/8
90000/90000 [==============================] - 792s 9ms/step - loss: 0.2795 - acc: 0.8787 - val_loss: 0.5031 - val_acc: 0.7842

Epoch 00006: loss improved from 0.30518 to 0.27951, saving model to C:\Users\lenovo\models\4cnn-06-0.280-0.879-0.503-0.784.hdf5
Epoch 7/8
90000/90000 [==============================] - 798s 9ms/step - loss: 0.2454 - acc: 0.8959 - val_loss: 0.5737 - val_acc: 0.7793

Epoch 00007: loss improved from 0.27951 to 0.24539, saving model to C:\Users\lenovo\models\4cnn-07-0.245-0.896-0.574-0.779.hdf5
Epoch 8/8
90000/90000 [==============================] - 795s 9ms/step - loss: 0.2332 - acc: 0.9005 - val_loss: 0.5837 - val_acc: 0.7761

Epoch 00008: loss improved from 0.24539 to 0.23316, saving model to C:\Users\lenovo\models\4cnn-08-0.233-0.900-0.584-0.776.hdf5
dict_keys(['val_loss', 'val_acc', 'loss', 'acc', 'lr'])
```

Figure 29. Accuracy achieved by 4 Conv – NN



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 40, 200)           18000200
_________________________________________________________________
dropout_1 (Dropout)          (None, 40, 200)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               168448
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
activation_1 (Activation)    (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 18,176,969
Trainable params: 18,176,969
Non-trainable params: 0
```

Figure 30. Params in LSTM Model



```
None
c:\users\lenovo\anaconda3\lib\site-packages\tensorflow_core\python\framework\indexed_slices.py:433: UserWarning: Converting sparse
known shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 90000 samples, validate on 10000 samples
Epoch 1/5
90000/90000 [==============================] - 361s 4ms/step - loss: 0.5163 - acc: 0.7474 - val_loss: 0.5033 - val_acc: 0.7594

Epoch 00001: loss improved from inf to 0.51634, saving model to C:\Users\lenovo\models\lstm-01-0.516-0.747-0.503-0.759.hdf5
Epoch 2/5
90000/90000 [==============================] - 357s 4ms/step - loss: 0.4543 - acc: 0.7881 - val_loss: 0.4650 - val_acc: 0.7781

Epoch 00002: loss improved from 0.51634 to 0.45425, saving model to C:\Users\lenovo\models\lstm-02-0.454-0.788-0.465-0.778.hdf5
Epoch 3/5
90000/90000 [==============================] - 357s 4ms/step - loss: 0.4150 - acc: 0.8113 - val_loss: 0.4625 - val_acc: 0.7887

Epoch 00003: loss improved from 0.45425 to 0.41504, saving model to C:\Users\lenovo\models\lstm-03-0.415-0.811-0.463-0.789.hdf5
Epoch 4/5
90000/90000 [==============================] - 355s 4ms/step - loss: 0.3764 - acc: 0.8326 - val_loss: 0.4646 - val_acc: 0.7838

Epoch 00004: loss improved from 0.41504 to 0.37638, saving model to C:\Users\lenovo\models\lstm-04-0.376-0.833-0.465-0.784.hdf5
Epoch 5/5
90000/90000 [==============================] - 356s 4ms/step - loss: 0.3424 - acc: 0.8502 - val_loss: 0.4946 - val_acc: 0.7791

Epoch 00005: loss improved from 0.37638 to 0.34240, saving model to C:\Users\lenovo\models\lstm-05-0.342-0.850-0.495-0.779.hdf5
```

Figure 31. Accuracy achieved by LSTM Model

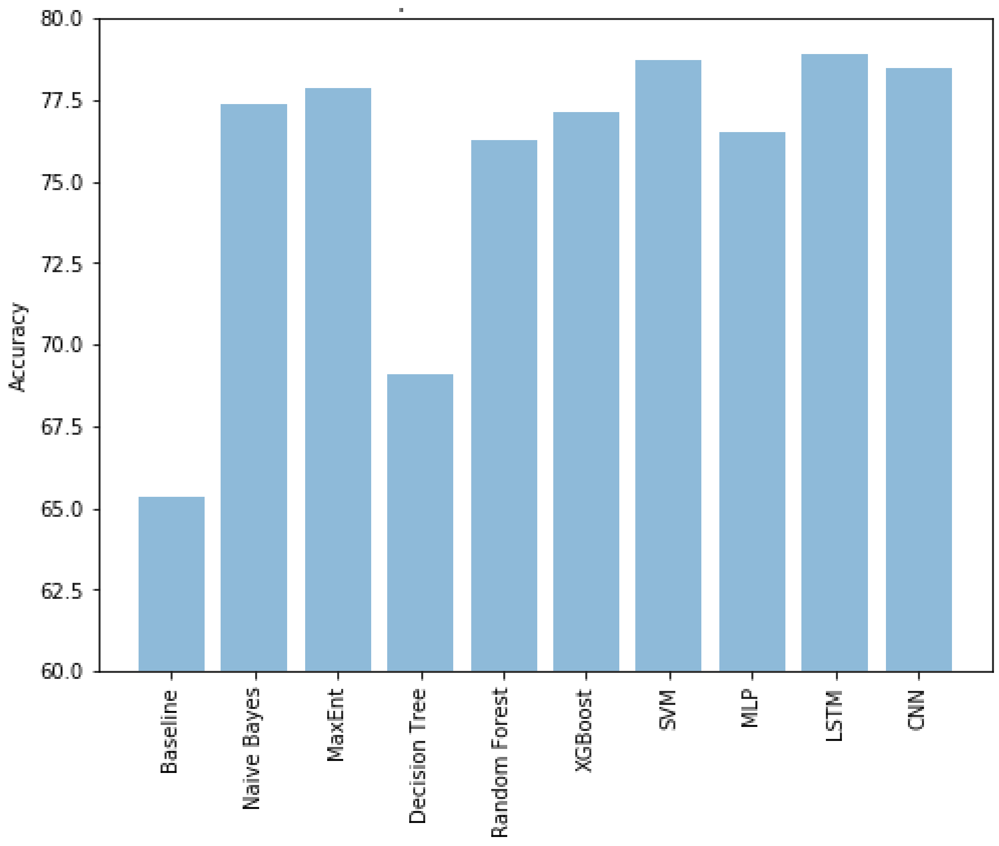


Figure 32. Comparison of various Classifications

CHAPTER 6

CONCLUSION

The provided tweets were a mixture of words, emoticons, URLs, hashtags, user mentions, and symbols. Before training the we pre-process the tweets to make it suitable for feeding into models. We implemented several machine learning algorithms like Naive Bayes, Maximum Entropy, Decision Tree, Random Forest, XGBoost, SVM, Multi-Layer Perceptron, Recurrent Neural networks and Convolutional Neural Networks to classify the polarity of the tweet. We 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.

Future directions

• **Handling emotion ranges:** We can improve and train our 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. We can therefore classify the sentiment in ranges, say from -2 to +2.

• **Using symbols:** During our pre-processing, we discard most of the symbols like commas, full-stops, and exclamation mark. These symbols may be helpful in assigning sentiment to a sentence.

REFERENCES

[1] A Agarwal, B Xie, I Vovsha, O Rambow, RJ Passonneau. (2011). Sentiment Analysis of Twitter Data. *Proceedings of the Workshop on Language in Social Media (LSM 2011),* 30-38

[2] V Kharde, P Sonawane (2016). Sentiment analysis of twitter data: a survey of technique *arXiv preprint arXiv:1601.06971*

[3] G Gautam, D Yadav, 2014, Sentiment analysis of twitter data using machine learning approaches and semantic analysis *Seventh International Conference on Contemporary Computing (IC3),* 437-442

[4] B Gokulakrishnan, P Priyanthan, T Ragavan, N Prasath, AS Perera (2012). Opinion mining and sentiment analysis on twitter data stream *Advances in ICT for Emerging Regions (ICTer), 2012 International Conference …*

[5] VS Pagolu, KN Reddy, G Panda, B Majhi (2016) Sentiment analysis of Twitter data for predicting stock market movements *international conference on signal processing, communication, power, and …*

[6] A Mittal, A Goel (2012) Stock prediction using twitter sentiment analysis *Stanford University, CS229 (2011 http://cs229. Stanford. edu/proj2011 …*

[7] SA Bahrainian, A Dengel (2013) Sentiment analysis and summarization of twitter data *2013 IEEE 16th International Conference on Computational Science and …*