Twitter Sentiment Analysis using Long Short-Term Memory (LSTM), Support Vector Machine and Logistic Regression

Durgashree Shetty, Rachana Ramesh, Meet Shah
San Diego State University.

Abstract*: Twitter is a micro-blogging website that allows Millions of users to share and express their views about topics, or post messages. Sentiment Analysis is the process of 'computationally' determining whether a piece of writing is positive, negative or neutral. Natural Language Processing (NLP) is a unique subset of Machine Learning which cares about the real-life unstructured data. The aim of this project is to show how to train and develop a simple Twitter Sentiment Analysis supervised learning models using python and NLP libraries and to develop a functional classifier for accurate and automatic sentiment classification of an unknown tweet stream.

I. Introduction

Sentiment Analysis is known as opinion mining, deriving the opinion or attitude of a speaker. Analyzing the public sentiment is important for many applications such as firms trying to find out the response of their products in the market, review of movie, company, book, predicting political elections and predicting socioeconomic phenomena like stock exchange.

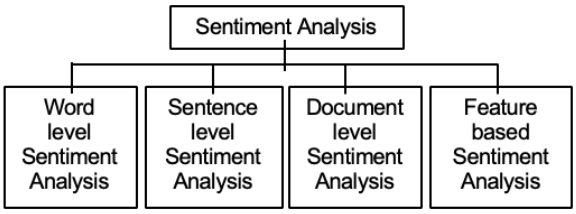

Fig 1.Level of Sentiment Analysis

Word Level: Dictionary based approaches; Corpus based approaches

Sentence Level: Sentence level sentiment classification classifies sentence into positive, negative or neutral class.

Document Level: It deals with tagging individual documents with their sentiment. It classifies either into positive or negative class i.e. Find the sentiment polarities of individual sentences or words and combine them together to find the polarity of the document.

Feature based: It deals with labeling each word with their sentiment and also identifying the entity towards which the sentiment is directed.

Natural Language Processing (NLP) is a unique subset of Machine Learning which cares about the real-life unstructured data. NLP is a hotbed of research in data science these days and one of the most common applications of NLP is sentiment analysis. Although computers cannot identify and process the string inputs, the libraries like NLTK, TextBlob and many others found a way to process string mathematically. Twitter is a platform where most of the people express their feelings towards the current context. As humans, we can guess the sentiment of a sentence whether it is positive or negative. Similarly, in this project we train and develop a simple Twitter Sentiment Analysis using different supervised learning model like Long Short-Term Memory (Recurrent Neural Network), Support Vector Machine and Logistic Regression.

II. Dataset

Our main objective of this project was to classify the tweets based on whether they propagated hate or not ie racist tweets vs non racist tweets. Our dataset consisted of total of 49159 tweets. It was then divided as 65% training set that had 31962 data values and 35% testing set of about 17197 data values. The dataset was obtained from a GitHub repository.

The dataset was thoroughly cleaned by removing twitter handles, punctuations, special symbols and numbers. It was then tokenised and normalised using nltk's PorterStemmer() function. After cleaning, the top hashtags were plotted and following plots were obtained:

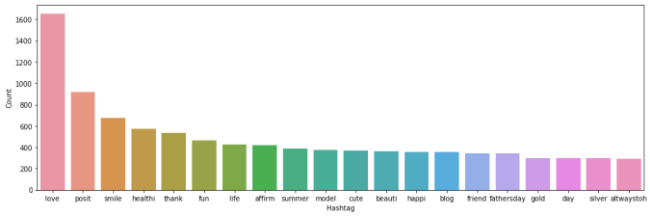

Fig 2. Plot of top hashtags of non-racist tweets

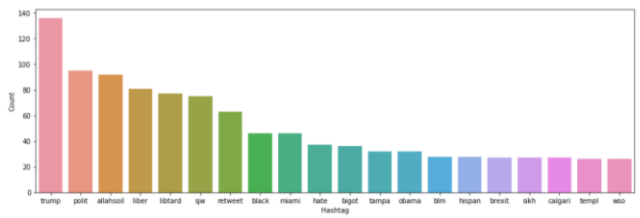

Fig 3. Plot of hashtags of racist tweets

III. Models Used

A. Long Short Term Memoty (Recurrent Neural Network):

RNN-LSTM for Twitter sentiment analysis. Unlike standard RNN, LSTM has feedback connection. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. LSTM networks are well-suited to classifying text data.

1) Preprocessing of data for LSTM RNN:

To prepare the data for the LSTM model first store all the tweets from csv file using pandas, using NLTK package remove all the English stop words from the reviews since stop words does not have any impact on classifying reviews. Get the list of appos to modify the reviews if it present. Remove all the special character from reviews using punctuation, once the reviews are cleaned get the count of each words from the dataset. Once we get the formatted reviews encodes review in to array of numbers. Here we took sequence length as 250, if the review int length is less than 250 then pad the sentence with zeros, else get the truncate to the input to sequence length. Below plot shows the total formatted review with the length of the review int, the most the review lies between 10 to 15 length. The count of is 31959. [1]

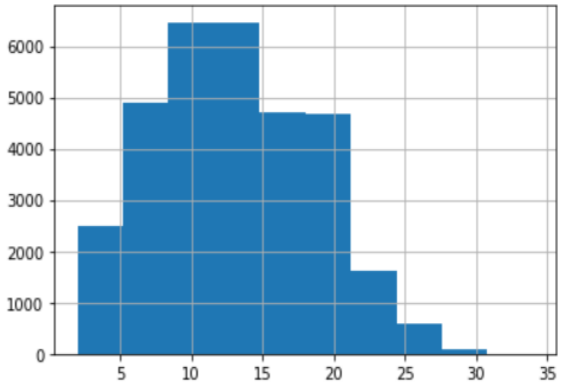

Fig 4. Formatted reviews with review int length

2) Design of the LSTM RNN model:

We have designed our LSTM-RNN using pytorch.[2] Our model is composed of 2 layers. We tested with one layer and two layers since as a general 1 hidden layer work with simple problems, like this, and two are enough to find reasonably complex features. We have used 3 epochs and 10 epochs. In our case, adding a second layer only improves the accuracy by ~0.1%. We have used Sigmoid function as activation function. The Embedding layer is used to create word vectors for incoming words. The output of the Embedding layer is the input to the LSTM layer. Embedding holds the dimension of vocab-size and 400, we have used pytorch embedding function to achieve embedding layer. LSTM layer we are taking 2 layer and drop out as 0.5, we are using pytorch LSTM function to calculate the LSTM layer. For dropout First we are talking 0.3 as probability of an element using pytorch dropout module. Linear layer with hidden dimension 256 is used first. Pytorch Linear module is used which applies a linear transformation to the incoming data: math: y = xA^T + b. The Training parameter are epochs 10 and gradient clipping is 5. Model is trained with mentioned data and calculated loss in every epoch. After validation data set, we passed model for Test Data to calculated the Test loss and Test Accuracy.
After calculating test accuracy, we generated rnn_submission.csv which labeled tweets with 0 or 1 for our test data. i.e the automatic sentiment classification of an unknown tweet stream is created.

B. Support Vector Machine:

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. In this algorithm, a plot each data item is done as a point in n-dimensional space (where n is number of features) with the value of each feature being the value of a particular coordinate. Then classification is done by finding the hyper-plane that differentiates the two classes correctly.
The objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that other data points can be rightly classified. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class.

$$(\theta) = C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_i^2$$

Also, the cost function is minimized. Cost, for SVMs, is determined by kernel (similarity) functions.
For this project, Linear kernel is used. It is used when the data is linearly separable, that is, it can be separated using a single line. The linear kernel is given as follows:

K (x, xi) = sum (x * xi)

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM. It is ideal to use because it gives faster results and better accuracy.
For this classification, label 0 is used to classify non-racist tweets and label 1 for racist tweets.[6]

1) Design of SVM model:

To begin with, the data of the dataset was cleaned by removing user handles, punctuations, symbols and other unwanted values. This was done by matching with regular expression and was then tokenized using the split () function.The stemporter () function from NLTK package removed all the suffixes from words. This was the process involved in cleaning of data.
Then, the features were extracted from sklearn.feature_extraction. CountVectorizer and TfidfVectorizer were imported and max_df=0.90, min_df=2, max_features=1000 were set for bag of words and tf-idf methods. The genism package was imported for Word2vec and size was set to 200 with window=5.
The data was then split into training and validation set.
To design the SVM model, svm package was imported from sklearn library as *from sklearn import svm.* The svm.svc model C-support vector classification is used for this project. C-Support Vector Classification implementation is based on libsvm. The fit time scales at least quadratically with the number of samples. The parameters for svm.svc are kernel=linear and regularisation parameter c=1(default). It was then fit into bow, tf-idf and w2vec. For the validation, the threshold was initially to 0.5 and later to 0.3 and the results after applying on test data are as mentioned at the end. The final results were displayed using F1 score and accuracy

score which were were imported from sklearn.metrics library. The prediction labels were submitted to a csv file.

Bag of Words, TF-IDF and word2vec techniques were used to perform the sentiment analysis.

1) Bag of Words: is a way of extracting features from text used mainly for machine learning algorithms.
A bag-of-words is a representation of text that describes the occurrence of words. This method is simple and flexible.
It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

The model is only concerned with whether known words occur in the document, not their order in the document.

2) TF-IDF: Term Frequency- Inverse Document Frequency: weight is a statistical measure for understanding the importance of a word in the corpus which increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

Term Frequency (TF): is a scoring of the frequency of the word in the corpus.

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

Inverse Document Frequency (IDF): is a scoring of how rare the word is across documents.

$$IDF(t) = log_e(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

TF-IDF score= TF * IDF

3) Word2Vec: takes a large corpus of text as input and assigns a vector for each unique word. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.[5]

C. Logistic Regression:

Logistic regression for sentiment analysis. Logistic regression in a way is very appropriate for these types of problems because Logistic regression is a statistical way of analyzing the dataset in which there are only two possible outcomes. Logistic Regression categorizes the output into two which can either be represented as a binary – 1/0 or True/False or some custom "label" as defined by the user. Sentiment analysis as we already know needs the output as "0" or "1" which stands for Positive and Negative sentiment of the statement. Logistic regression predicts the probability of an event by fitting the data into a special sigmoid function. g(z) = 1/(1+e^−z), is called the logistic function or the sigmoid function. The sigmoid function gives us a value that ranges between 0 to 1. We then set a threshold value for that value to belonging to either of the class. If the value of g(z) is less than the threshold then it as belonging to class 0 else if it greater than the threshold it will belong to class 1. The threshold value is set by the user as per the strictness of output. [7]

We use two methods to classify the sentiment of any text as Positive or Negative. Bag of words and TF-IDF method.

1) Bag of words method: It is one of the simplest methods for sentiment analysis. Here every word is counted for the number of occurrences it appears in the data collection. Once the count is ready, we take individual words into account and assign a score to them. If the total score is negative then the text will be classified as a negative and if it is positive then the text is classified as positive. Steps for Bag of words method are:

a) Create a list of words.

b) Extract every single word from the sentence in the file and tokenize the word.

c) Store every word on the list created.

d) Use a counter to count the occurrence of each word in the dataset.

The words that you found out in the bag-of-words will now act as the feature. All the feature table rows are independent feature vectors containing information about a specific 'tweet', the particular words and its sentiment. Note that, the label sentiment is 0 or 1. Also, it is not necessary that all the 'tweet's will contain each of these features/words. Now we simply pass these features to the classifier and fit the LogisticRegression() method to this feature table.

2) TF–IDF: it stands for Term Frequency-Inverse Document Frequency. As it is made up of two terms Term Frequency and Inverse Document Frequency, we will talk about both of them. Term frequency is defined as the ratio:

$$TF(x,d) = \frac{number\ of\ times\ term(x) appears\ in\ data\ (d)}{total\ number\ of\ terms\ in\ data(d)}$$

Now we compute the inverse of this ratio as:

$$IDF = log\left(\frac{total\ number\ of\ terms\ in\ data(d)}{number\ of\ times\ term(x) appears\ in\ data\ (d)}\right)$$

We can calculate the resulting TF-IDF by simply multiplying the TF with the IDF for each element. TF-IDF is the same as the count vectorizer, TFIDF vectorizer will calculate these scores for terms in documents, and convert textual data into the numeric form. Later we fit the LogisticRegression() method to this feature table obtained. [8]

IV. RESULTS

A) *Displaying the outputs for LSTM:*

On testing model with layer 1 and 2 the difference was ~0.1%. One hidden layer work with simple problems, like this, and two are enough to find reasonably complex features. In our case, adding a second layer only improves the accuracy by ~0.1% (94.399 vs. 94.432). Also 3-4 epoch is approx. where we noticed the validation loss stop decreasing. Even the one-layer work while generating output csv we have used two layers.

```
Epoch: 1/3... Step: 100... Loss: 0.288736... Val Loss: 0.238306
Epoch: 1/3... Step: 200... Loss: 0.157442... Val Loss: 0.187071
Epoch: 1/3... Step: 300... Loss: 0.142676... Val Loss: 0.153595
.....
```

Fig 5. Output of loss and val loss for with 100 steps

The training loss and accuracy for 1 layer and 2 layers for 3 and 10 epochs are shown in below table:

epochs	n-layer	Loss	Accuracy
3	1	0.256	94.399
3	2	0.189	94.432
10	1	0.48	93.807
10	2	0.459	93.901

Table 1. LSTM RNN outputs

B) Displaying the outputs for SVM:

SVM is imported from sklearn and after running svm.SVC function for linear kernel by fitting for the above-mentioned techniques, the outputs are as follows:

Threshold>=0.3

Bag of Words F1 score is 0.50831792976
Bag of Words Accuracy is 0.94451976222
TF-IDF F1 score is 0.510483135825
TF-IDF Accuracy is 0.943998331421
Word to Vector F1 score is 0.600778210117
Word to Vector Accuracy is 0.946501199291

Threshold>=0.5

Bag of Words F1 score is 0.4683026584867075
Bag of Words Accuracy is 0.9457711961622692
TF-IDF F1 score is 0.47960199004975124
TF-IDF Accuracy is 0.94545833767859
Word to Vector F1 score is 0.5405405405405406
Word to Vector Accuracy is 0.9485869225153822

C) Displaying the outputs for Logistic Regression:

Using Bag of Words method:
Logistic Regression F1 score using Bag of Words method is: 0.482621648461
Logistic Regression Accuracy using Bag of Words method is: 0.945666910001
Which is equivalent to: 94.57 %
Bag of Words Training time: 1576381565.793829s
Bag of Words Prediction time: 0.004995s

Using TF-TDF method:
Logistic Regression F1 Score using TF-IDF method is: 0.458773784355
Logistic Regression prediction using TF-IDF method is: 0.946605485452
Which is equivalent to: 94.66 %
TF-IDF Training time: 1576381567.286654s and Prediction time: 0.004994s

As we can see that the accuracy for Bag of Words is not accurate as TF-IDF as it does not take the word order and grammar into account. Bag of words focuses on the words that appear a greater number of times in the dataset whereas TF-IDF focuses more on the words that appear lesser compared to the other words. The words having lesser frequency have a higher value rather than a lower value. Hence the more impactful words are sometimes neglected by the Bag of word method. Hence the accuracy of TF-IDF is better compared to Bag of Words. The threshold value for this observation is 0.5.

V. Link To Code

We have added the repository to GitHub: https://github.com/makshah/CS596-Project-Sentiment-Analysis.git

VI. Author Contributions

Durgashree Shetty implemented the LSTM(RNN) model. Rachana Ramesh implemented SVM model. Meet Shah implemented Logistic Regression model. All three contributed equally to prepare the report. All authors were deeply involved in the project and provided critical feedback that helped to build a successful project.

VII. Hardware and Software Specifications

All models are tested on Nvidia GTX 970M with CUDA 10.1 under Windows OS. We used Anaconda Jupyter Notebook as the IDE.

VIII. Conclusion

Based on the results of the implemented model, it can be concluded that all the models produced almost the same accuracy. In LSTM(RNN), as the number of hidden layers were increased, the accuracy also increased. In SVM, word2vec technique gave the highest F1 score and accuracy as compared to the other two. In Logistic Regression, the TF-IDF method gave a good F1 score and accuracy. Thus, the results were quite consistent for the given dataset.

Model		Accuracy
LSTM-RNN	1-Layer	94.39%
	2-Layer	94.43%
SVM	Bag of Words	94.57%
	TF-IDF	94.54%
	Word to Vector	94.85%
Logistic Regression	Bag of Words	94.57%
	TF-IDF	94.66%

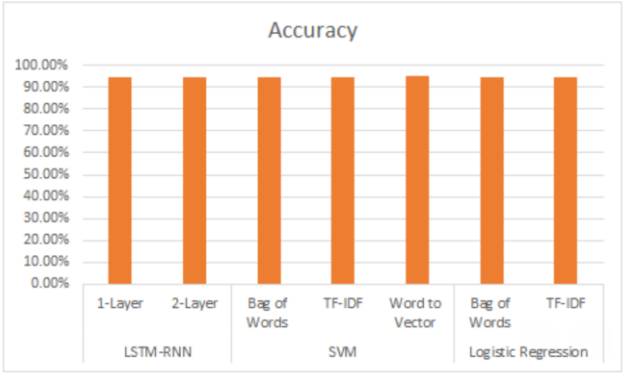

Fig. 5 Combined outputs of all models

IX. Future Scope

The accuracy we received for the Sentiment Analysis project is around 94% for every model hence it can be used reliably to create applications such as news headline analysis to make an analysis whether the total news of a particular day. We can also try to implement an impact factor for the

predictions so that we can get the positivity value impact factor. In cases where there is some grave bad news and some minor good news so the original count will show that the day is neutral but in reality, it is not that good news. Sentiment analysis can also be used in defense cases where we can monitor someone's behavior by the basis of his or her tweets or messages to identify if the person is a potential threat. Furthermore, sentiment analysis can be used to check if any product or movie is good or bad based on the reviews by the users. There is an enormous number of applications where this can be used.

REFERENCES

[1] kaggle SentimentalAnalysis by Aarya

[2] Tutorial on Sentimental Analysis using Pytorch by Bhadresh: https://medium.com/@bhadreshpsavani/tutorial-on-sentimental-analysis-using-pytorch-b1431306a2d7

[3] Kaggle Twitter SentimentalAnalysis by Alireza: https://www.kaggle.com/arkhoshghalb/twitter-sentiment-analysis-hatred-speech

[4] Dataset:https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

[5] SVM:https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[6] Logistic Regression: https://slundberg.github.io/shap/notebooks/linear_explainer/Sentiment%20Analysis%20with%20Logistic%20Regression.html

[7] Bag of words, TF-IDF: https://itnext.io/machine-learning-sentiment-analysis-of-movie-reviews-using-logisticregression-62e9622b4532