Sentiment Analysis

Jos Katiyare
Analytics Club
IIT Bombay
Mumbai, India
joshkatiyare1903@gmail.com

***Abstract*——Social networks are the main resources to gather information about people's opinion and sentiments towards different topics as they spend hours daily on social medias and share their opinion. In this technical paper, we show the application of sentimental analysis and how to connect to Twitter and run sentimental analysis queries. We run experiments on different queries from politics to humanity and show the interesting results. We realized that the neutral sentiment for tweets are significantly high which clearly shows the limitations of the current works.**

***Index Terms*—Twitter sentiment analysis, Social Network analysis**

I. Introduction

Opinion and sentimental mining is an important research areas because due to the huge number of daily posts on social networks, extracting people's opinion is a challenging task. About 90 percent of today's data has been provided during the last two years and getting insight into this large scale data is not trivial [17, 18].

Sentimental analysis has many applications for different domains for example in businesses to get feedbacks for products by which companies can learn users's feedback and reviews on social medias.

Opinion and sentimental mining has been well studied in this reference and all different approaches and research fields have been discussed [10]. sentimental analysis however in this paper we mostly focus on the Twitter sentimental analysis.

Twitter nowadays is one of the popular social media which according to the statistain [4] currently has over 300 millions accounts. Twitter is the rich source to learn about people's opion and sentimental analysis [2]. For each tweet it is important to determine the sentiment of the tweet whether is it positive, negative, or neutral. Another challenge with twitter is only 140 characters is the limitaiton of each tweet which cause people to use phrases and works which are not in language processing. Recently twitter has extended the text limitations to 280 characters per each tweet.

II. Literature Survey

Some works have used an ontology to understand the text [1]. In the phrase level, sentimental analysis system should be able to recognize the polarity of the phrase which is discussed by Wilson, et.al [2]. Tree kernel and feature based model have been applied for sentimental analysis in twitter by Agarwal and et.al [3]. SemEval-2017 [4] also shows the seven years of sentimental analysis in twitter tasks. Since tweets in Twitter is a specific text not like a normal text there are some works that address this issue like the work for short informal texts [5]. Sentimental analysis has many applications in news [6].

In this paper, we will discuss social network analysis and the importance of it, then we discuss Twitter as a rich resource for sentimental analysis. In the following sections, we show the high-level abstract of our implementation. We will show some queries on different topics and show the polarity of tweets.

III. DATASETS

Since social networks, especially Twitter, contains small texts and people may use different words and abbreviations which are difficult to extract their sentiment by current Natural Language processing sysntems easily, therefore some researchers have used deep learning and machine learning techniques to extract and mine the polarity of the text [15]. Some of the top abbreviations are FB for facebook, B4 for before, OMG for oh my god and so on. Therefore sentimental analysis for short texts like Twitter's posts is challengeing [8].

IV. ANALYSIS PIPELINE

The approach to extract sentiment from tweets is as follows:

- Start with downloading and caching the sentiment dictionary
- Download twitter testing data sets, input it in to the program.
- Clean the tweets by removing the stop words.
- Tokenize each word in the dataset and feed in to the program.
- For each word, compare it with positive sentiments and negative sentiments word in the dictionary. Then increment positive count or negative count
- Finally, based on the positive count and negative count, we can get result percentage about sentiment to decide the polarity.

After getting the data many tasks are performed of it to clean it and make it easy to generate desirable results. As described above these tasks includes lemmetization, tokenization, stemming, url removal, lower casing etc. were done with the help of nltk package.

Once our data is cleaned we can move ahead to do some basic analysis by building word clouds, analysing length etc. Few results from those tasks are listed below:

1) *Word cloud for Positive Dataset*

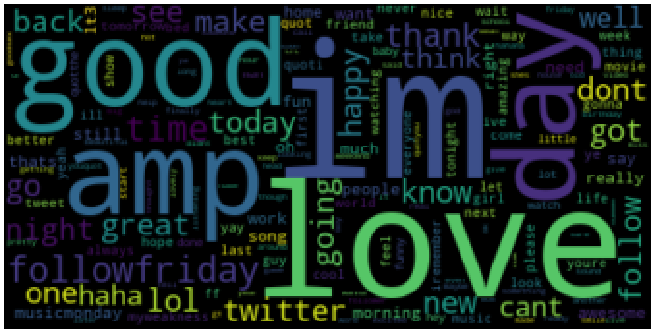


Fig. 1. Word cloud for Positive Dataset

2) *Word cloud for negative Dataset*

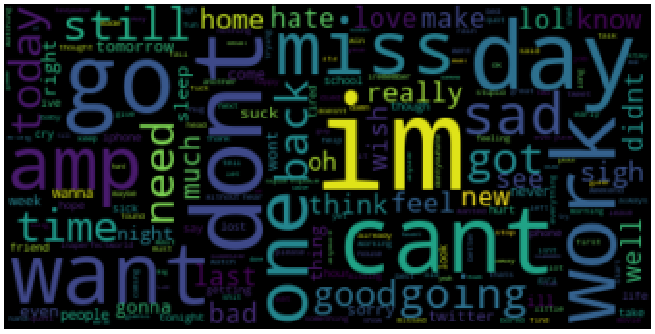


Fig. 2. Word cloud for negative Dataset

3) *Length of tweets*

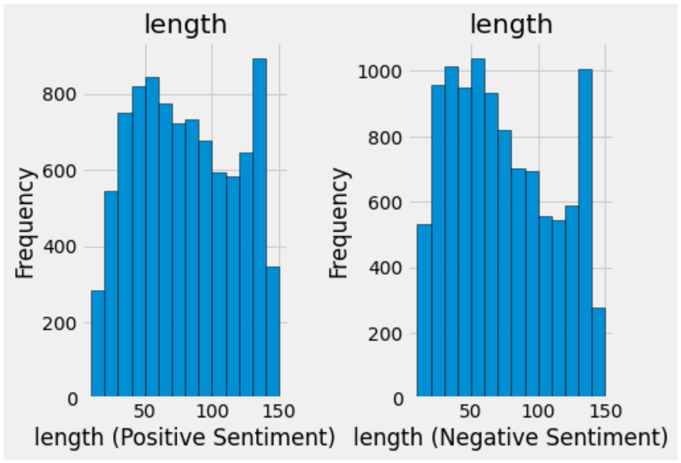


Fig. 3. Length of tweets

V. RESULTS

In this section, the paper presents machine learning tasks our dataset. In the first we implemented lstm and in the second task we also incorporated word2vec.

A. Creating Word embedding

1) *Word2Vec* Word2Vec was developed by Google and is one of the most popular technique to learn word embeddings using shallow neural network. Word2Vec can create word embeddings using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW).

Word2Vec() function creates and trains the word embeddings using the data passed.
Training Parameters

- size: The number of dimensions (N) that the Word2Vec maps the words onto. Bigger size values require more training data, but can lead to better (more accurate) models.
- workers: Specifies the number of worker threads for training parallelization, to speed up training.
- min_count: min_count is for pruning the internal dictionary. Words that appear only once or twice in a billion-word corpus are probably uninteresting typos and garbage. In addition, there's not enough data to make any meaningful training on those words, so it's best to ignore them.

2) *Tokenizing and Padding datasets* Tokenization is a common task in Natural Language Processing (NLP). It's a fundamental step in both traditional NLP methods like Count Vectorizer and Advanced Deep Learning-based architectures like Transformers.
Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. Hence, tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.
All the neural networks require to have inputs that have the same shape and size. However, when we pre-process and use the texts as inputs for our model e.g. LSTM, not all the sentences have the same length. We need to have the inputs with the same size, this is where the padding is necessary.
Padding is the process by which we can add padding tokens at the start or end of a sentence to increase it's length upto the required size. If required, we can also drop some words to reduce to the specified length.
Tokenizer: Tokenizes the dataset into a list of tokens. pad_sequences: Pads the tokenized data to a certain length. The input_length has been set to 60. This will be the length after the data is tokenized and padded.
Tokenizing the X_train and X_test dataset and padding them to the length 'input_length'.
The tokenized list is pre-padded, i.e padding tokens are added to the start. After padding, the length of the data would be equal to 'input_length'.

3) *Word Embedding:* Word Embedding is a representation of text where words that have the same meaning have a

similar representation. In other words it represents words in a coordinate system where related words, based on a corpus of relationships, are placed closer together. In the deep learning frameworks such as TensorFlow, Keras, this part is usually handled by an embedding layer which stores a lookup table to map the words represented by numeric indexes to their dense vector representations.
Embedding Matrix is a maxtrix of all words and their corresponding embeddings. We use embedding matrix in an Embedding layer in our model to embedded a token into it's vector representation, that contains information regarding that token or word.
We get the embedding vocabulary from the tokenizer and the corresponding vectors from the Embedding Model, which in this case is the Word2Vec model. Shape of Embedding matrix is usually the Vocab Length * Embedding Dimension.

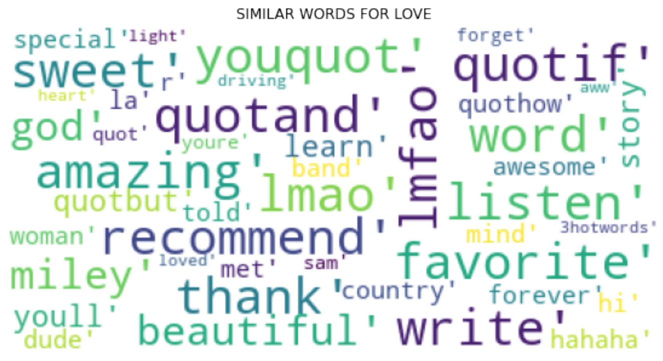


Fig. 4. Similar words for love using Word2Vec

B. Layers

1) *Deep network:* Deep network takes the sequence of embedding vectors as input and converts them to a compressed representation. The compressed representation effectively captures all the information in the sequence of words in the text. The deep netwrok part is usually an RNN or some forms of it like LSTM. The dropout is added to overcome the tendency to overfit, a very common problem with RNN based networks.
2) *Fully Connected Layer:* The fully connected layer takes the deep representation from the LSTM and transforms it into the final output classes or class scores. This component is comprised of fully connected layers along with batch normalization and optionally dropout layers for regularization.
3) *Output Layer:* Based on the problem at hand, this layer can have either Sigmoid for binary classification or Softmax for both binary and multi classification output.

C. Model Architecture

1) *Embedding Layer:* Layer responsible for converting the tokens into their vector representation that is generated by Word2Vec model. We're using the predefined layer from Tensorflow in out model.
Arguments -
   - input_dim: Size of the vocabulary.
   - output_dim: Dimension of the dense embedding.
   - weights: Initiazises the embedding matrix. trainable: Specifies whether the layer is trainable or not.
2) *LSTM:* Long Short Term Memory, its a variant of RNN which has memory state cell to learn the context of words which are at further along the text to carry contextual meaning rather than just neighbouring words as in case of RNN.
Arguments -
   - units: Positive integer, dimensionality of the output space.
   - dropout: Fraction of the units to drop for the linear transformation of the inputs.
   - return_sequence: Whether to return the last output in the output sequence, or the full sequence.
3) *Dense:* Dense layer adds a fully connected layer in the model. The argument passed specifies the number of nodes in that layer.
The last dense layer has the activation "Sigmoid", which is used to transform the input to a number between 0 and 1. Sigmoid activations are generally used when we have 2 categories to output in.

D. Model Callback

Callbacks are objects that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc).

We can use callbacks to write TensorBoard logs after every batch of training, periodically save our model, stop training early or even to get a view on internal states and statistics during training.

ReduceLROnPlateau: Reduces Learning Rate whenever the gain in performance metric specified stops improving.

- *monitor:* quantity to be monitored.
- *patience:* number of epochs with no improvement after which learning rate will be reduced.
- *cooldown:* number of epochs to wait before resuming normal operation after lr has been reduced.

EarlyStopping: Stop training when a monitored metric has stopped improving.

- monitor: Quantity to be monitored.
- min_delta: Minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta, will count as no improvement.
- patience: Number of epochs with no improvement after which training will be stopped.

E. Model Compile

The Model must be compiled to define the loss, metrics and optimizer. Defining the proper loss and metric is essential while training the model.

- Loss: We're using Binary Crossentropy. It is used when we have binary output categories. Check out this article on losses.

- Metric: We've selected Accuracy as it is one of the common evaluation metrics in classification problems when the category data is equal. Learn more about metrics here.
- Optimizer: We're using Adam, optimization algorithm for Gradient Descent

We'll now train our model using the fit method and store the output learning parameters in history, which can be used to plot out the learning curve.

Arguements:

- batch_size: Number of samples per gradient update. Increasing the batch_size speeds up the training.
- epochs: Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided.
- validation_split: Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch
- callbacks: List of callbacks to apply during training process.
- verbose: 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.

F. *Evaluating Model*

1) *The Learning curve:* Learning curves show the relationship between training set size and your chosen evaluation metric (e.g. RMSE, accuracy, etc.) on your training and validation sets. They can be an extremely useful tool when diagnosing your model performance, as they can tell you whether your model is suffering from bias or variance.

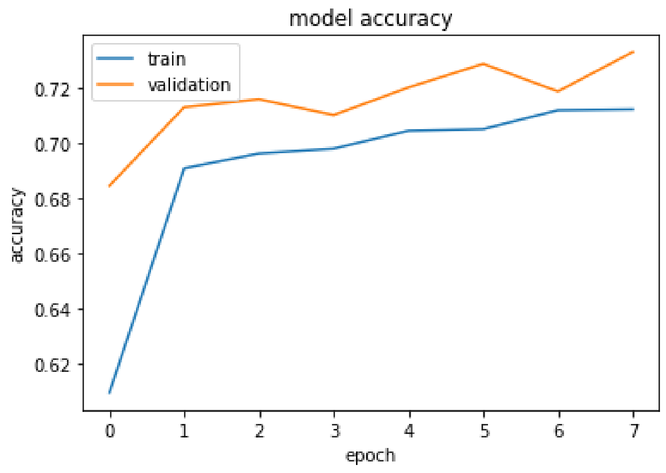


Fig. 5. Model Accuracy

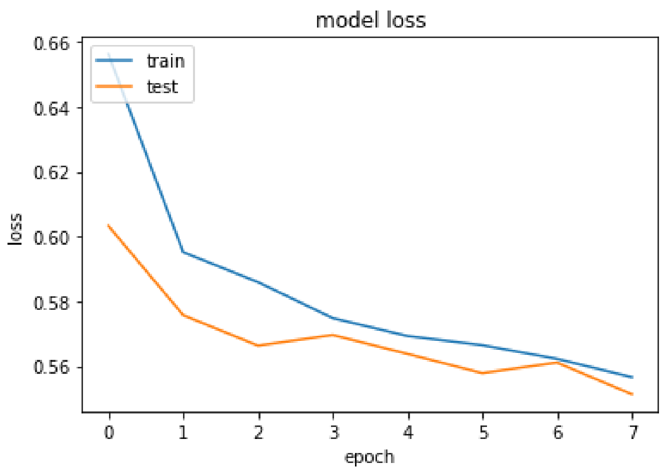


Fig. 6. Model Loss

2) *Confusion Matrix:*

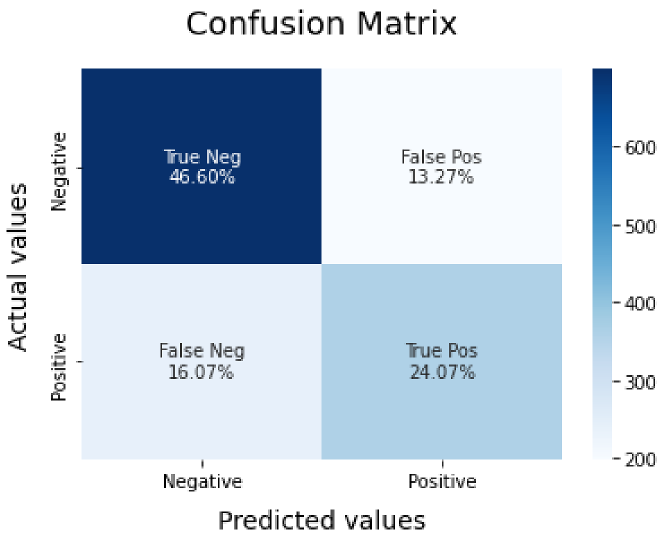
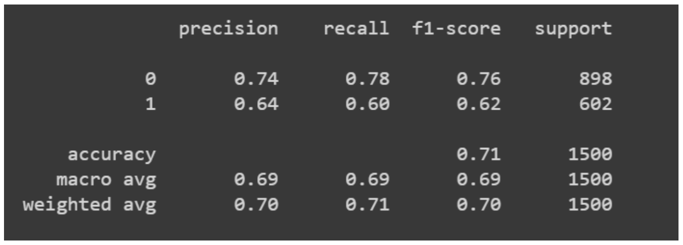


Fig. 7. Confusion Matrix

3) *Classification Report:* Using the classification report, we can see that that the model achieves nearly 71% Accuracy after training for just 8 epochs. This is really good and better than most other models achieve



```
              precision    recall  f1-score   support

           0       0.74      0.78      0.76       898
           1       0.64      0.60      0.62       602

    accuracy                           0.71      1500
   macro avg       0.69      0.69      0.69      1500
weighted avg       0.70      0.71      0.70      1500
```

Fig. 8. Classification Report

VI. Discussion

In this report we have implemented sentiment analysis on tweets using Word2Vec and LSTM. First we started with cleaning our dat using lemmetization, removal of stopwords, removal of urls, tokenization etc. Once our data is set we implemented Word2Vec. Using that we learned word embedding. It helped in capturing context of the word. And after that we created embedding matrix. LAter we used embedding matrix in our model building. Our model architecture comprises of embedding layer, LSTM layer and dense layer. And then we compiled our model using Adam optimizer.

The accuracy of this model is come out to be 73 % which is not that bad. It is also an improvement from the logistic regression base model, we used for comparison.

VII. Conclusion and Future Scope

In this technical paper, we discussed the importance of social network analysis and its applications in different areas. We focused on Twitter as and have implemented the python program to implement sentimental analysis. We realized that the neutral network worked quite perfectly but there is always a room for improvement. And building a better model architecture we can significantly improve our accuracy.

References

[1] Boguslavsky, I. (2017). Semantic Descriptions for a Text Understanding System. In Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference “Dialogue”(2017) (pp. 14-28)

[2] Wilson, T., Wiebe, J., Hoffmann, P. (2005, October). Recognizing contextual polarity in phrase-level sentiment analysis. In Proceedings of the conference on human language technology and empirical methods in natural language processing(pp. 347-354). Association for Computational Linguistics.

[3] Agarwal, A., Xie, B., Vovsha, I., Rambow, O., Passonneau, R. (2011, June). Sentiment analysis of twitter data. In Proceedings of the workshop on languages in social media (pp. 30-38). Association for Computational Linguistics.

[4] Rosenthal, S., Farra, N., Nakov, P. (2017). SemEval-2017 task 4: Sentiment analysis in Twitter. In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)(pp. 502-518).

[5] Kiritchenko, S., Zhu, X., Mohammad, S. M. (2014). Sentiment analysis of short informal texts. Journal of Artificial Intelligence Research, 50, 723-762.

[6] Balahur, A., Steinberger, R., Kabadjov, M., Zavarella, V., Van Der Goot, E., Halkia, M., ... Belyaeva, J. (2013). Sentiment analysis in the news. arXiv preprint arXiv:1309.6202

[7] Sentiment Analysis
url = https://www.kaggle.com/haresrv/sentiment-analysis

[8] LSTM Sentiment Analysis — Keras
url = https://www.kaggle.com/ywang311/lstm-sentiment-analysis-keras

[9] Text Preprocessing in NLP with Python codes
url = https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/