***Social Media Sentiment Analysis***

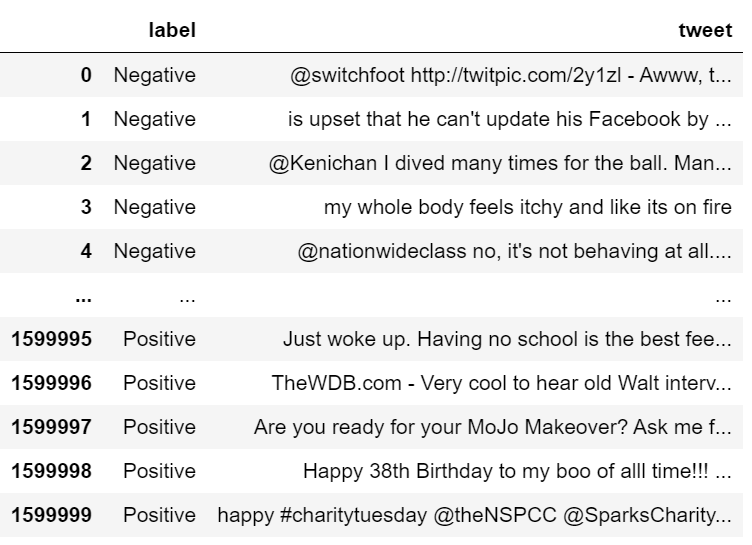
by Shreyash Gaur

1. INTRODUCTION

Sentiment analysis, a sub-field of Natural Language Processing, is one of the most popular topics and research fields in data science. I will be working on social media sentiment analysis. I aim to be able to classify tweets, reviews and comments from social media as positive, negative or neutral. The most important point of our project is data mining to collect a large amount of data from several sources. For this purpose, I found open source datasets such as Sentiment140 and many others. After all the searching I decided to use the Sentiment140. Most of the open-source datasets that I found on the internet are properly labeled and structured. Data collected by ourselves needs to be properly labeled. Then, I will go through the cleaning, preprocessing and separation of test and training data steps. I searched for some tools for our project and found some popular and powerful open-source NLP frameworks in Python. I will probably use the Natural Language Toolkit (NLTK). It comes with all the pieces you need to get started on sentiment analysis.

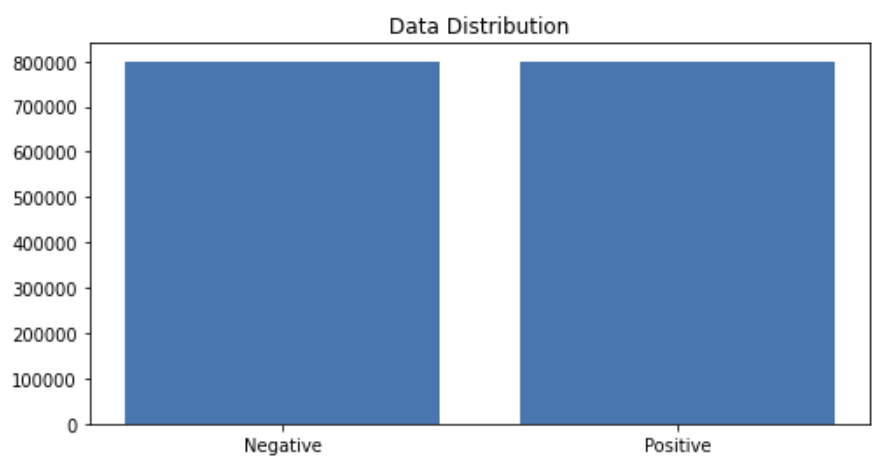
1. APPROACH

First of all, I have to apply preprocesses to our dataset to avoid unexpected results. Our dataset contains 1,600,000 tweets extracted using the Twitter API. The tweets have been classified from 0 (negative) to 4 (positive). The dataset contains 6 fields which are target as integer, ids as integer, date as date, flag as string, user as string and text as string.But I don’t need to use all of these fields. For our purpose, I eliminated 4 fields which don't serve our purpose. After this elimination, our dataset has only two fields, which are label and tweet. The updated dataset is shown in Figure 1.



*Figure 1. The Sentiment140 dataset after preprocess*

Finally, I removed the missing values from all dataset, and our dataset distribution after all transformations is shown in Figure 2.



*Figure 2. Data distribution of preprocessed dataset*

I can train the embedding ourselves. However, that approach can take a long time to train. So, I use transfer learning techniques, and I use GloVe: Global Vectors for Word Representation. The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford. It is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. I downloaded the GloVe. Then, I initialize an embedding index that has 400000 word vectors, and an embedding matrix. I chose to use LSTM, CNN and Multinomial Naive Bayes algorithms for classification/regression.

1. EXPERIMENT SETUP

First of all, I apply preprocessing to data and analysis in detail. In the preprocessing part, I applied data reduction and cleared stop words and punctuations from all instances. Then, I analyzed the data in terms of letter frequencies, distribution of the letters relative to the expected frequency of English language with chi-square test, word frequencies and their maximum, minimum and standard deviation. Then, I have analyzed the most common words in 2 classes that are positive and negative instances. Lastly, I used feature extraction methods, bag-of-words, and word embedding. Bag-of-words with TF-IDF is a common and simple way of feature extraction. I have created and analyzed correlation of words in corpus with this way.

For classification/regression experiments, the test set percentage is set to be 20%. 6 different models that are applied are CNN Model-1 with 1024 batch size, CNN Model-2 with 512 batch size, LSTM Model-1 with 1024 batch size, LSTM Model-2 with 512 batch size, Multinomial Naive Bayes Model-1 with Count vectorizer and Multinomial Naive Bayes Model-2 with TF-IDF vectorizer. I have chosen precision, recall, f1-score, AUC and ROC to evaluate our models.

1. EXPERIMENTAL RESULTS AND DISCUSSION
2. Preprocessing

In the preprocessing part of the project, I mainly have analyzed the data in 2 terms, which are letter and word. Firstly, By counting the letters of the instances, I have analyzed frequency and relative frequency of the letters of the whole dataset. Then, I applied the chi square test to see whether the distribution of the letters in data is the same as what I expect from English texts.

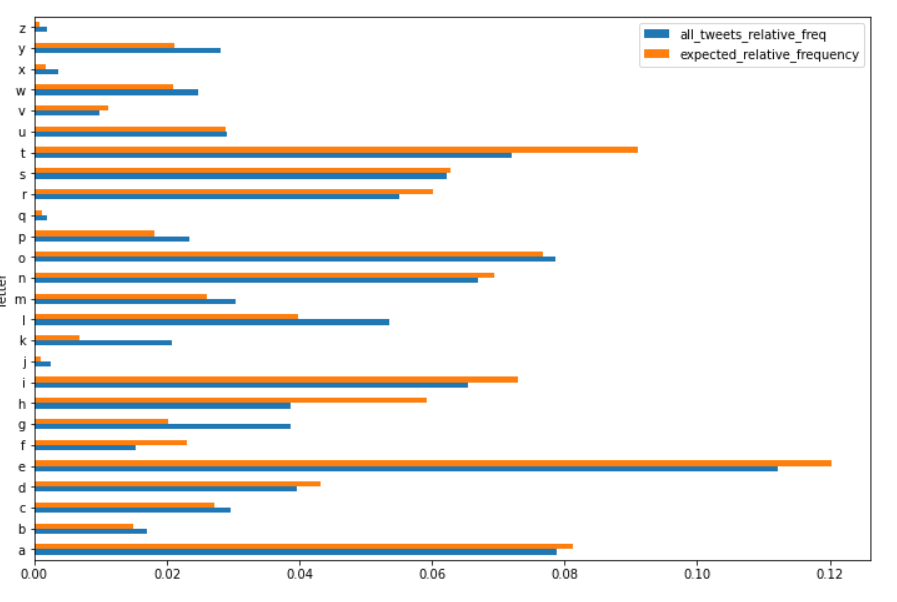


Figure 3. Letter frequencies of each 26 characters in English Alphabet.



Figure 4. Letter frequency of the dataset, relative frequencies of the dataset, expected relative frequency according to the English language and expected character length according to the English language.

Then, I got the p-value (p) as 0 which implies that the letter frequency does not follow the same distribution with what I see in English tests, although the Pearson correlation is too high (~96.65%) as shown in Figure 6.

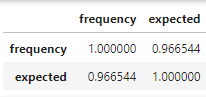


Figure 5. Correlation.

I counted the number of characters for each tweet and analyzed the data frame according to maximum number of characters, minimum number of characters, mean of the number of characters column and its standard deviation. Our longest tweet is 189 characters long, the shortest tweet is 1 character long and mean of all tweets’ character length 42.78. The standard deviation of all tweet character length is 24.16.

Secondly, I counted the number of words for each tweet and analyzed the data frame according to maximum number of words, minimum number of words, mean of the number of words column and its standard deviation. Our longest tweet is 50 words long, the shortest tweet is 1 word long and the mean of all tweets’ word length is 7.24. The standard deviation of all tweet character length is 4.03.

Also, I have analyzed the most common words in 2 classes that are positive and negative instances.

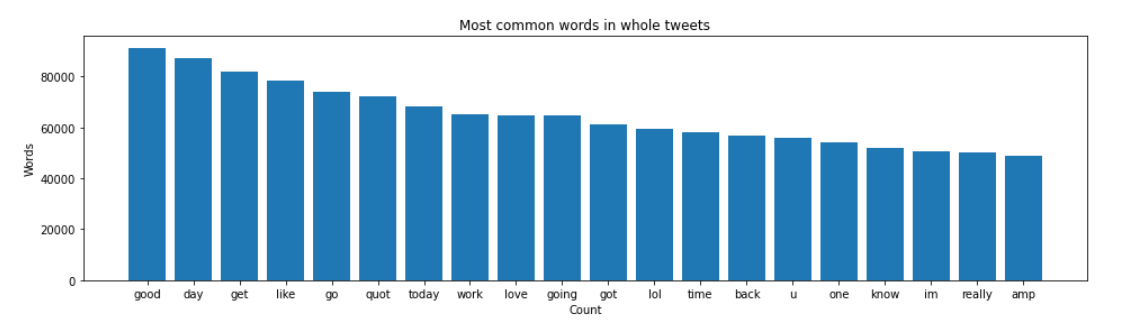


Figure 6. Most common words in our dataset.

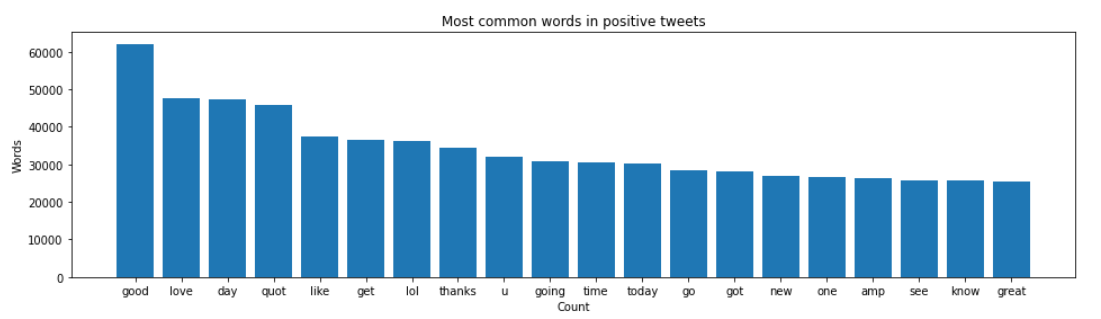


Figure 7. Distribution of most common words in positive tweets in our dataset*.*



Figure 8. Most common words in positive tweets in our dataset.

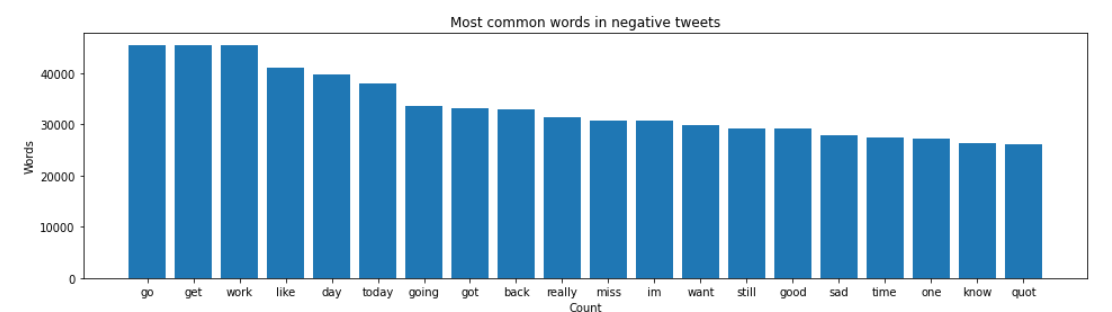


Figure 9. Distribution of most common words in negative tweets in our dataset*.*



Figure 10. Most common words in negative tweets in our dataset*.*

I used feature extraction methods, bag-of-words, and word embedding. Bag of words with TF-IDF is a common and simple way of feature extraction. Bag-of-Words is a representation model of text data and TF-IDF is a calculation method to score the importance of words in a document.

After applying bag-of-words with TF-IDF, I create the scatter plot according to these results.

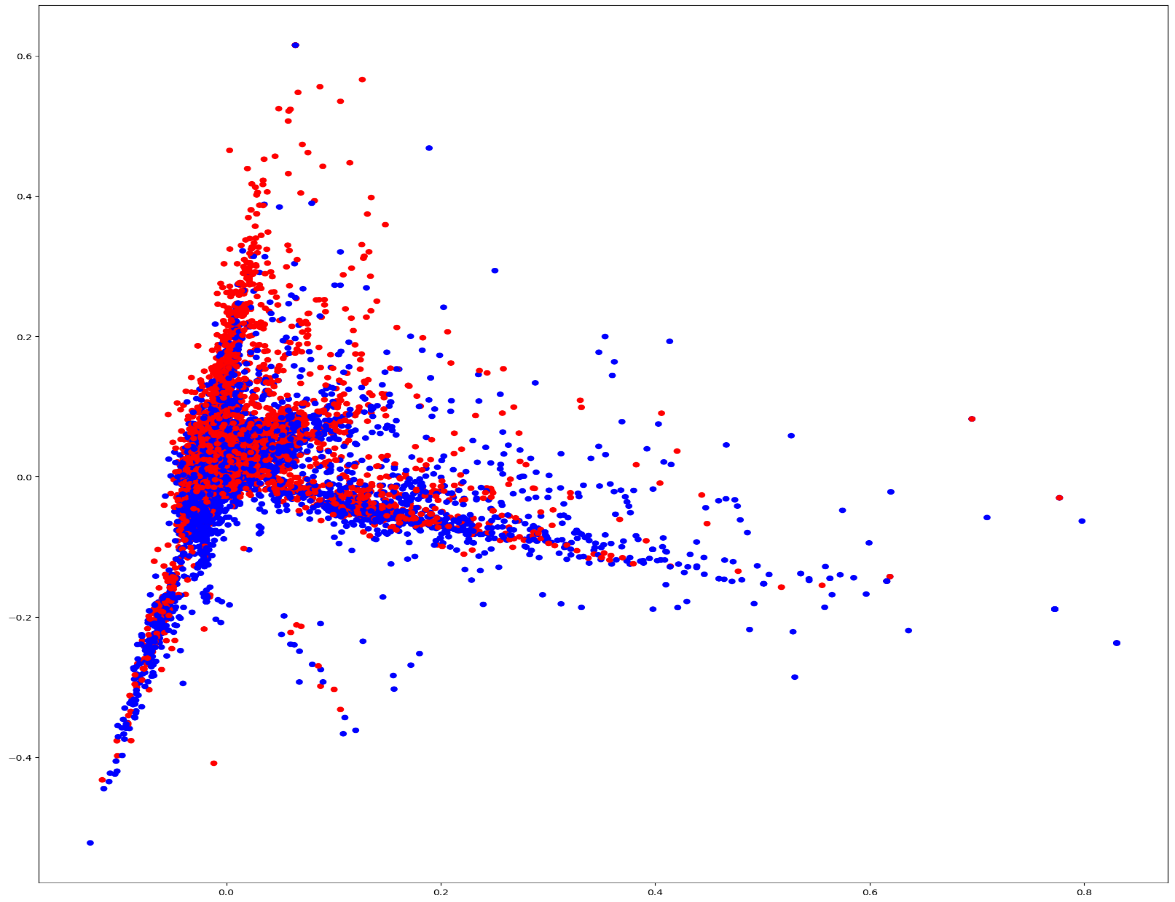


Figure 11. Scatter plot that shows correlation of words in the corpus: red indicates negatives, blue indicates positives.

1. Predictive Analysis

At the beginning, our dataset had 6 features which were target, id, date, query, user and text. I chose two of them for our purposes which are target and text. I can see that the entropy decreases significantly after this transformation.

Information gain

First entropy of dataset = 41.082

Entropy after preprocess = 14.733

For classification/regression experiments, the test set percentage is set to be 20%. 6 different models that are applied are CNN Model-1, CNN Model-2, LSTM Model-1, LSTM Model-2, Naive Bayes Model-1 and Naive Bayes Model-2.

CNN Model - 1 :

Conv1D = 64

Dense = 512

Dense = 512

1024 batch size

CNN Model - 2 :

Conv1D = 31

Dense = 256

Dense = 256

512 batch size

LSTM Model - 1 :

1024 Batch size

LSTM Model - 2 :

512 Batch size

Multinomial Naive Bayes Model - 1 :

Count Vectorizer

Multinomial Naive Bayes Model - 2 :

TF-IDF Vectorizer

Precision, recall, f1 score and accuracy of the models are shown below.

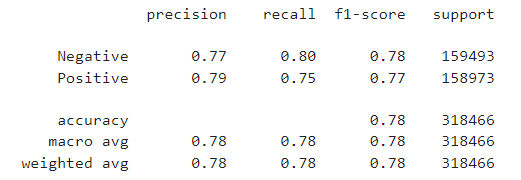


Figure 12. Precision, recall, f1 score and accuracy of the CNN Model-1

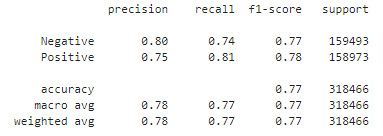


Figure 13. Precision, recall, f1 score and accuracy of the CNN Model-2

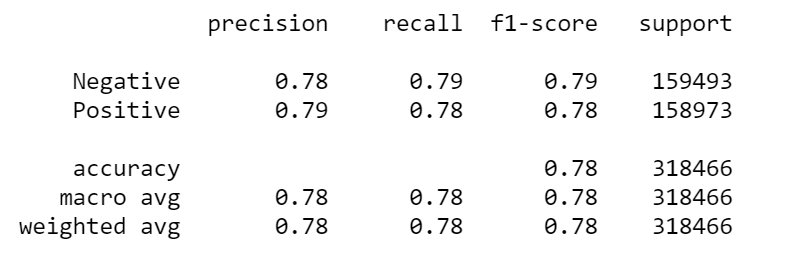


Figure 14. Precision, recall, f1 score and accuracy of the LSTM Model-1

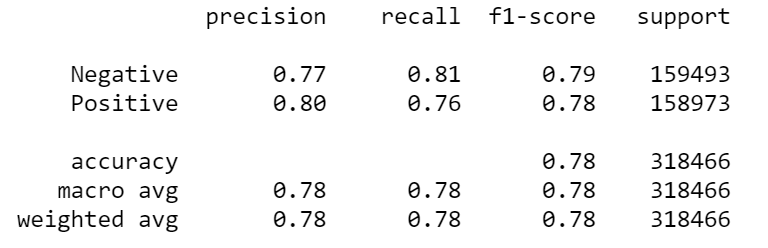


Figure 15. Precision, recall, f1 score and accuracy of the LSTM Model-2

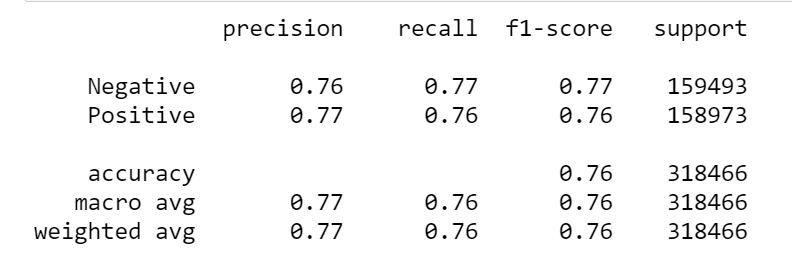


Figure 16. Precision, recall, f1 score and accuracy of the Multinomial Naive Bayes Model - 1 (CountVectorizer)

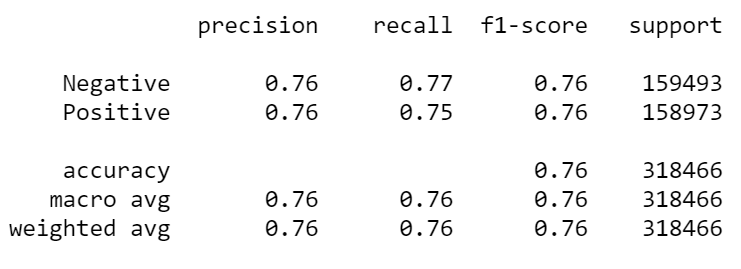


Figure 17. Precision, recall, f1 score and accuracy of the Multinomial Naive Bayes Model - 2 (TF-IDF Vectorizer)

After determining the evaluation metrics, ROC curves of the models are formed. Also, AUC values are calculated and shown at the bottom of each graph.

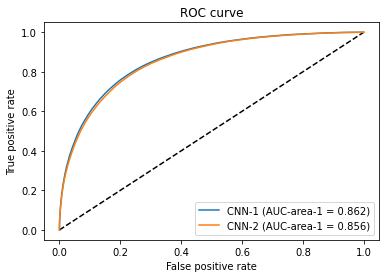


Figure 18. ROC Curve of CNN Model-1 and CNN Model-2



Figure 19. ROC Curve of LSTM Model-1 and LSTM Model-2.

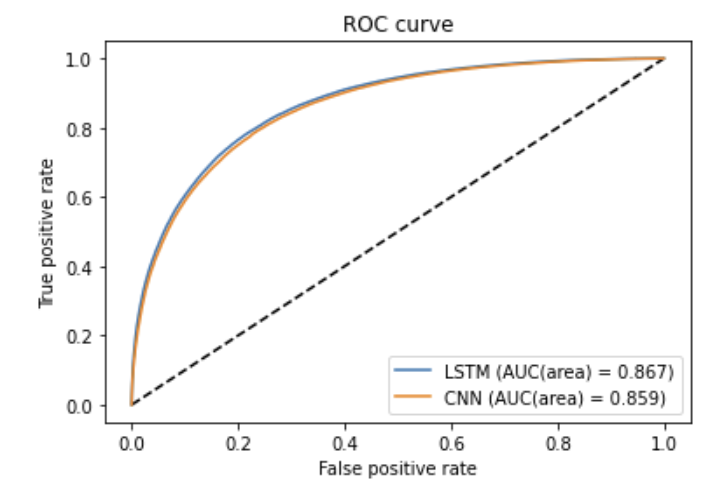


Figure 20. ROC Curve of best LSTM model and best CNN model.

Confusion matrices of the 6 model used to train the data, including the best performing model LSTM-1, are as follows:

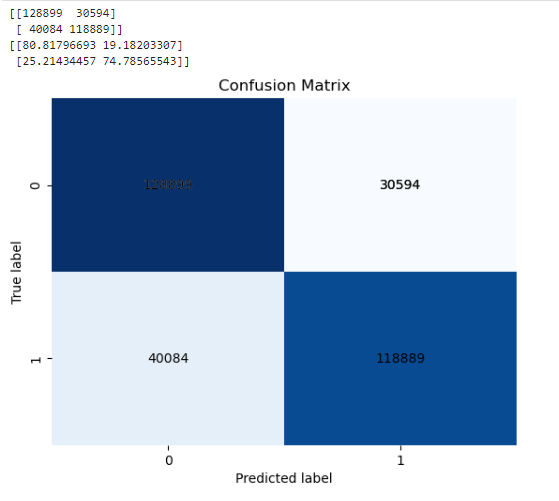


Figure 21. Confusion Matrix of CNN Model - 1.

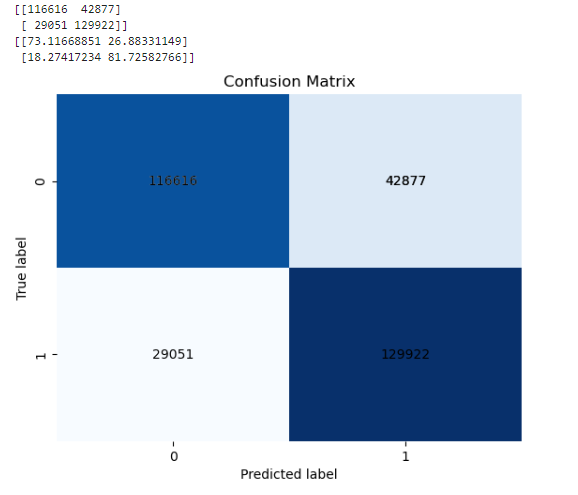


Figure 22. Confusion Matrix of CNN Model - 2.

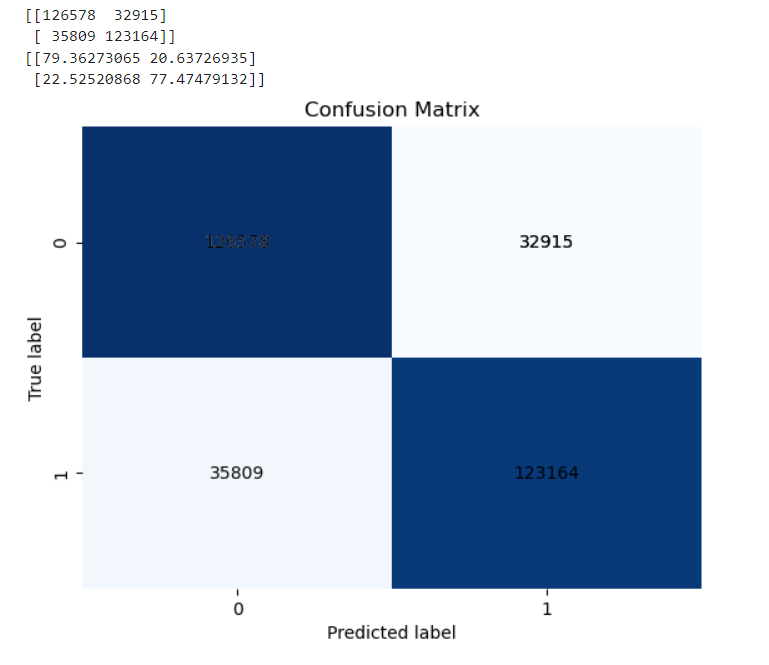


Figure 23. Confusion Matrix of LSTM Model - 1.

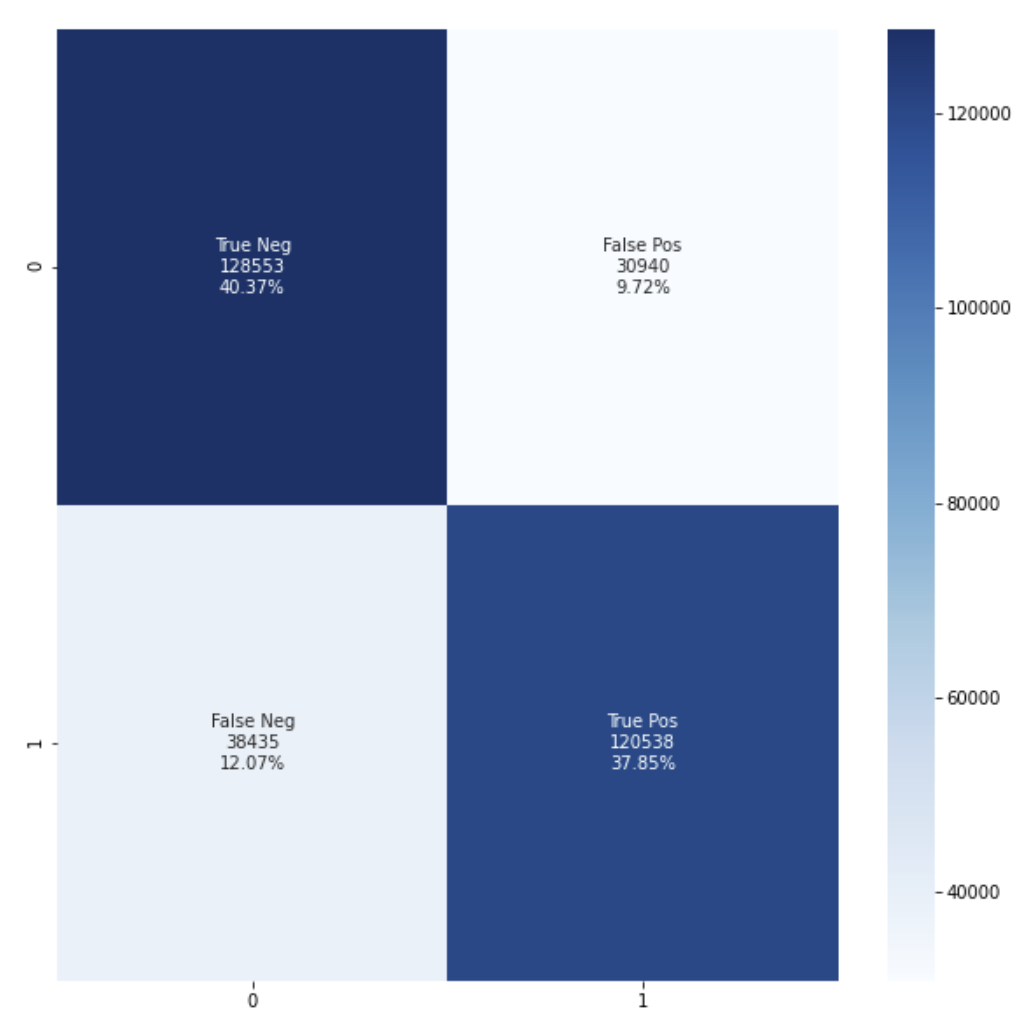


Figure 24. Confusion Matrix of LSTM Model - 2.

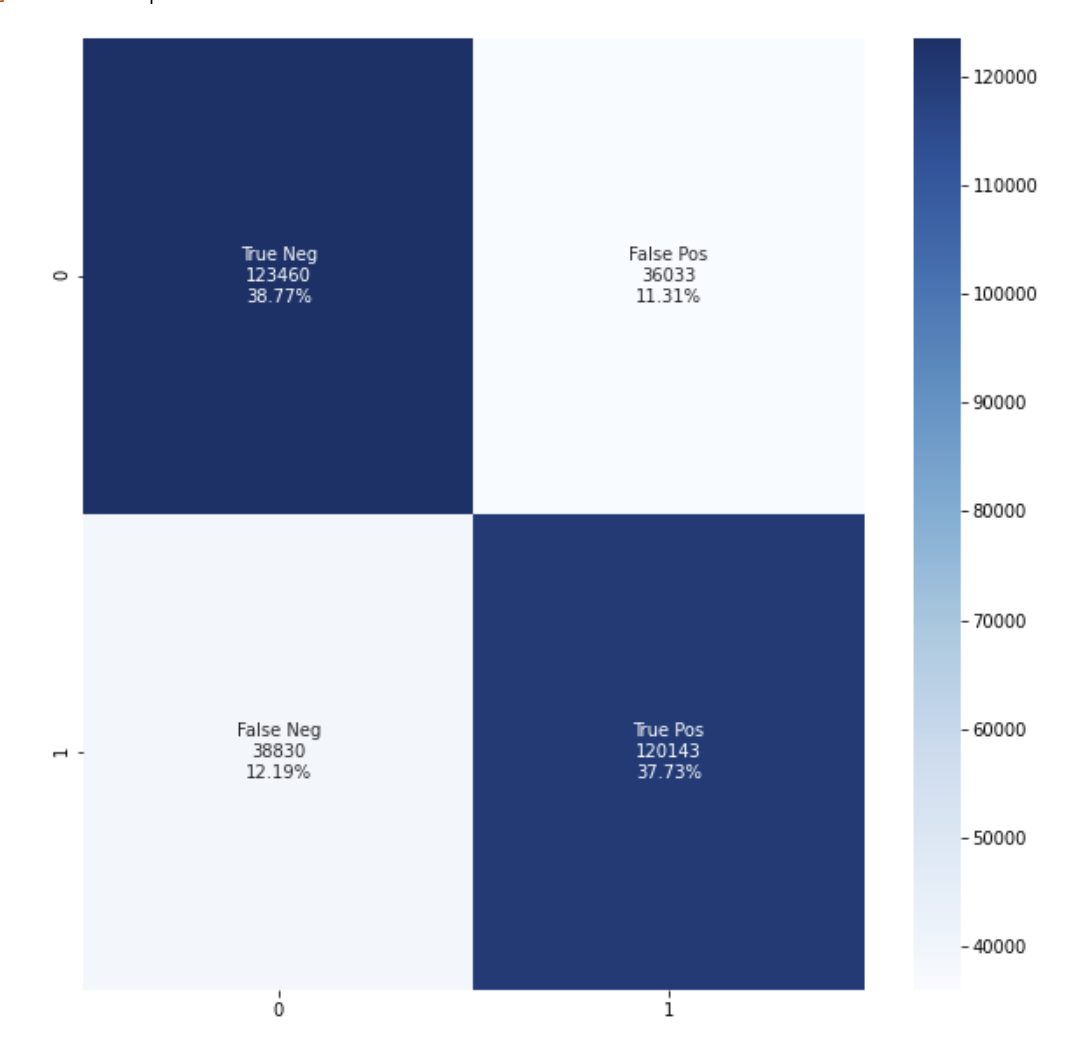


Figure 25. Confusion Matrix of Multinomial Naive Bayes with Count Vectorizer.

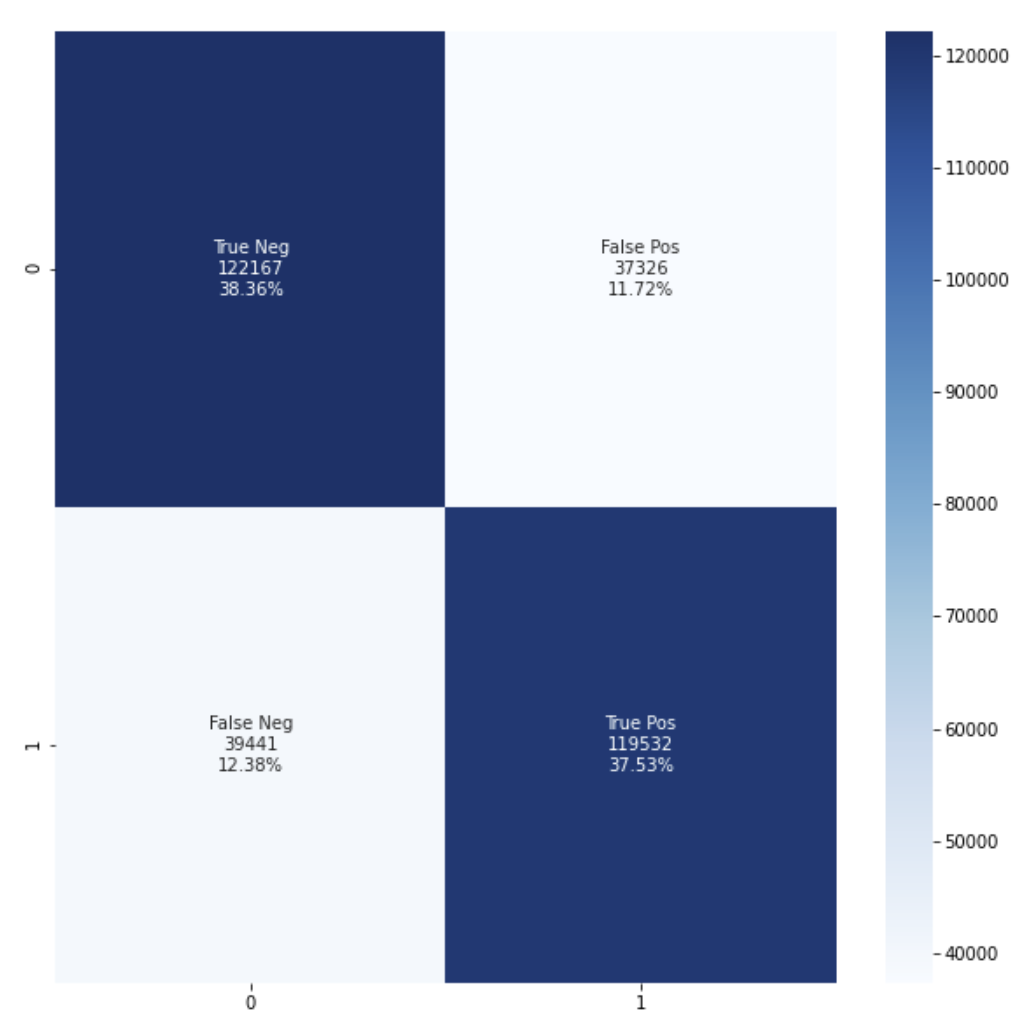


Figure 26. Confusion Matrix of Multinomial Naive Bayes with TF-IDF Vectorizer.

According to Accuracy, P, R, F1, AUC, our best performing model is LSTM model 1 with 1024 batch size and 0.789 accuracy and the closest competitor to LSTM model 1 is CNN model 1 with accuracy 0.781. Multinomial Naive Bayes with tf-idf is the worst performing algorithm among them, with accuracy 0.758.

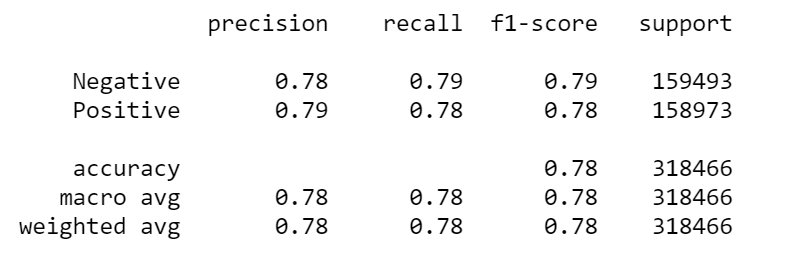


Figure 27. Accuracy, P, R and F1 of LSTM Model-1.

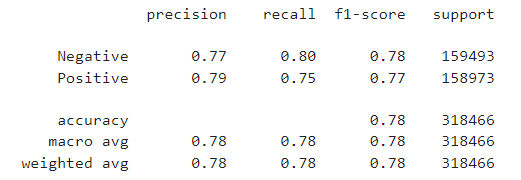


Figure 28. Accuracy, P, R and F1 of CNN Model-1.

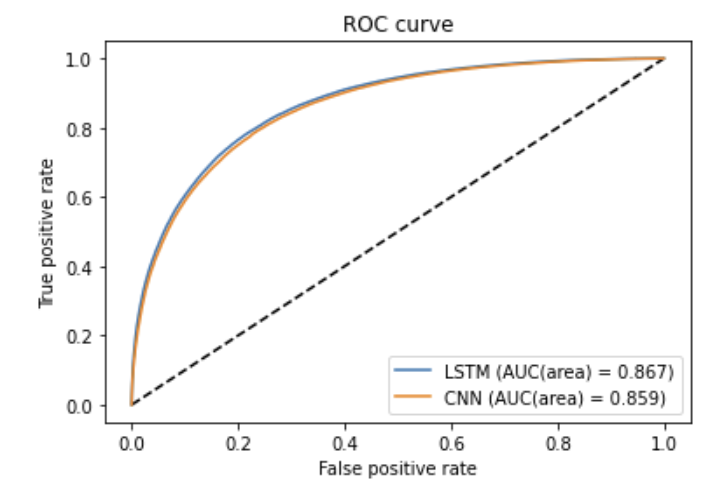


Figure 29. ROC - AUC Analysis of Best Performing Models.

1. CONCLUSION

Our raw dataset has unnecessary features for our purpose. Its first entropy value was 41.08. Then I dropped the unnecessary columns, deleted the empty valued rows, and I have obtained an entropy value of 14.73. After this preprocess, I can easily see that there is an important change in entropy values.

After all six experiments, I can see that different LSTM and CNN give us very close accuracy ratios after training. Although there are really low differences, LSTM Model-1 has the best result and Naive Bayes models performed slightly worse. Naive Bayes models have the best training time durations. It has very good speed compared to LSTM and CNN models. LSTM model-1, LSTM model-2 have close training times as each epoch takes 10 to 15 minutes for these models. Although changing the batch size in LSTM did not give an effective result difference, CNN model-2 has a better training time like 45 sec for each epoch. Also, its accuracy is really close to the others. LSTM model-1 has 78.9% accuracy rate with 1024 batch size and LSTM model-2 has 78.6% accuracy rate with 512 batch size. CNN model-1 has 78.2% accuracy rate with 1024 batch size and CNN model-2 has 77.2% accuracy rate with 512 batch size. Both algorithms have better training times with 512 batch size, are better than their 1024 batch sized models and their accuracy rates are really close. As a result of these, I can say that LSTM and CNN models with 1024 batch size are better for accuracy rate. But, models with 512 batch size have close accuracy rates within better training times.

For accuracy rates of Naive Bayes models there is a small difference like 1.5%. As a result of that, I can say that Naive Bayes with the CountVectorizer method gives better results than Naive Bayes with the TF-IDF method.