Detecting Depression using Sentiment Analysis on Tweets

Palak Goel	Rishita Chauhan	Ishita Kapur
2019438	2019383	2019364
IIIT-Delhi	IIIT-Delhi	IIIT-Delhi

1. Introduction

Nowadays, Social media is the platform where people can express and share their feelings with everyone. **Twitter** is one the most popular social media platforms where people express their views and hence help us to understand human behaviour. We can obtain the data from it and can predict the mental health problems of an individual. Depression is the common mental illness worldwide which may also lead to suicide. Depression is a very important subject to be discussed as it is a mental illness that affects most of the world's population. It's important that depression is detected in the initial stage so that the person can overcome it otherwise it may further lead to many mental disorders. It will help to create awareness in people regarding depression.

Sentiments expressed through the tweets gives us the concept of the deeper feelings of the users. We can extract the emotions and opinions from the tweets posted by the users and can predict whether the user is suffering from depression or not. In this project, we aim to predict depression through user's tweets using sentiment analysis.

2. Related Work

[4] Depression detection from social network data using machine learning techniques - PMC aims at analyzing online data of facebook, considering four different factors like emotional process, temporal process, linguistic style, and has used machine learning techniques like decision tree, k-nearest neighbor, svm, ensemble etc for detection. This paper reflects that decision trees gives better outcome.

[3] Depression Detection Using Machine Learning Techniques on Twitter Data — IEEE Conference Publication aims at detecting depression by using the data that the user shares on twitter (social media) and then two different classifiers are used to predict the results, and they are naive bayes and a hybrid model, NB tree. The results of this study show that, both the techniques are equally accurate, with an accuracy of 97.31% with 3000 tweets and 92.34% with 1000 tweets.

[2] Detecting Depression Through Tweets - Diveesh Singh aims at predicting depression among the twitter users based on their tweets. The machine learning techniques that were used are rnn, cnn and gru. Here the best performance were given by gru with 98% and cnn with 97%.

3. Dataset and Evaluation

3.1. Dataset

Dataset that we have used in this project is a combination of random tweets (non-depressive) [1] acquired from the kaggle website and depressive tweets extracted using twint tool. To extract depressive tweets, we have used some keywords that are specific to depression like Suicide, Depressed, Depression, Lonely, Hopeless and Antidepressant.

It consists of 3 columns x 18880 rows. The columns are id (tweet ID), tweet (text), and label (1 = depressive and 0 = non-depressive). Out of 18,880 samples, 8000 samples belong to non-depressive class and 10880 samples to depressive class.

For Baseline model (Logistic Regression): We splitted the dataset into 90% training (16992 samples) and 10% testing (1888 samples) data where we have performed K-fold (k=10) cross validation technique on the training dataset. For feature extraction, we have used the following techniques:
1. CountVectorization - It vectorizes the text into a sparse matrix representing all words in the text along with its occurrence.
2. Tf-IDF Normalization - It scales the features obtained from CountVectorization of the text into floating - point values indicating the relevance of the word in the entire dataset.
By performing CountVectorization and Tf-IDF Normalization technique, 2342 most relevant features were extracted.

For Advance model (RNN, LSTM, and CNN): Same testing set has been used as in the baseline model. While the training dataset is divided into training, and validation set in 80:10 ratio with 15292, and 1700 samples in respective sets. Further, we converted our dataset into Tensorflow

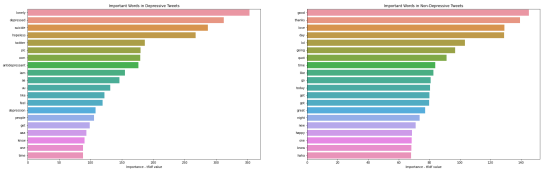
dataset with (text, label) pairs followed by the processing of new dataset using TextVectorization layer which converts the input text to a sequence of token indices which is then fed into the embedding layer.

3.2. Evaluation Metrics

To evaluate the performance of our model, we used metrics like Accuracy, Recall, Precision, F1 score, and Confusion matrix. AUC-ROC curve has also been used to assess the performance of the model. In order to check whether the model is underfitting or overfitting, we plotted the accuracy curve showing both training and validation accuracy. Loss curve was plotted during the training phase which gave us an idea of the direction in which the network learnt.

4. Methodology

4.1. Data Preprocessing



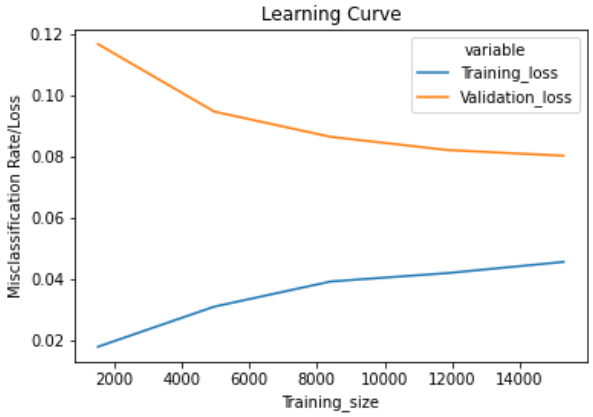
Raw text extracted from twitter contains unwanted characters and words which need to be removed before applying any classification technique to it. Hence, in NLP, text preprocessing is the first and an essential step in the process of building a model. We sanitized the tweets so that they do not contain any irrelevant text by removing URL links, and non-ASCII characters. Contractions were expanded, text converted to lowercase, @ mentions were removed along with other special characters, numbers, extra spaces and stopwords followed by Lemmatization of the text.

4.2. Learning Techniques Implemented

Baseline model: The baseline model was implemented using Sklearn's inbuilt Logistic Regression classifier with hyperparameters:

Regularisation C = 10.0, max_iter = 100, penalty = l2, solver = liblinear

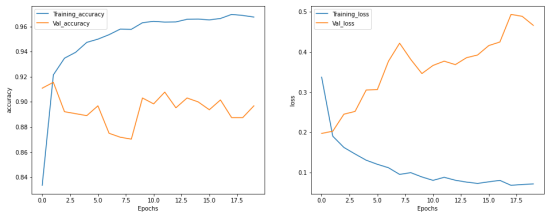
We used scikit-learn's GridSearchCV for hyperparameter tuning in order to learn the best hyperparameters on the given dataset, which were then used to initialize the final model and evaluate the test set. K-fold cross validation was used for model selection where we took k = 10. Selecting the appropriate number of features extracted using CountVectorizer and TfidfTransformer was the most challenging part in training the baseline model. It is not an effective practice to transform the whole vocabulary to features as there might be some rare words in the data corpus, which, if passed to TfidfVectorizer().fit(), will add unwanted dimensions to the inputs. The method we used to decide an optimal value for it included calculating the frequency of all the words across the corpus and then deciding on the words which occur frequently using a threshold value, which we took as 10. Hence, we considered only those words that occurred more than or equal to 10 times in the entire dataset while extracting the features. Through this technique, we ended up with 2342 features.



SimpleRNN: Once our baseline model was established, the first model we implemented was simple RNN. The very first layer of our model is the textVectorization layer which converts the input text to a sequence of token indices which is then fed into the embedding layer. "mask_zero" parameter of the embedding layer was set to True in order to handle variable length sequences. The final dense layer was fed through a dropout layer, with dropout_rate parameter set to 0.5, which used the sigmoid function from which the prediction was made. Binary Cross-Entropy loss has been used to compute loss between the predicted and true labels, which is minimized during training using Adam Optimizer. For hyperparameter tuning, keras_tuner has been used to learn the best hyperparameters values for the learning_rate and units of a dense layer, which were then used to initialize the final model and evaluate the test set. Hyperparameters used are as follows:

learning_rate = 0.001, dropout_rate = 0.5, batch_size = 32, epochs = 6

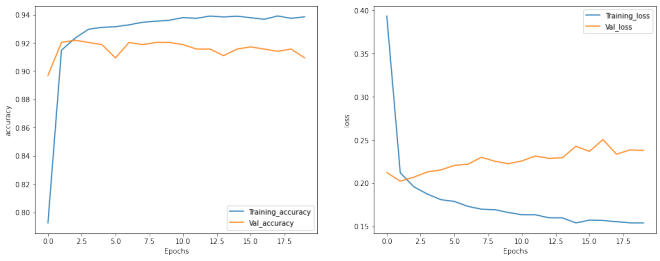
From the curve below, we can see that the Simple RNN model is fitting just right for the data when the number of epochs is less but tends to over fit when the number of epochs increase.



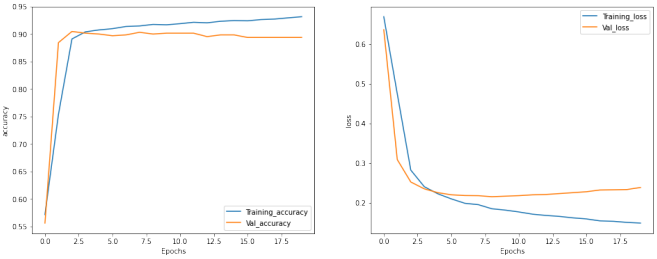
In fig, it can be seen that at epochs = 6 validation loss is the lowest. Hence, we have set epochs hyperparameter to 6.

LSTM: Next, we implemented the LSTM model. The first layer of the LSTM model is the textVectorization layer which converts the input text to a sequence of token indices. This was then fed into the embedding layer. The hidden dense layer was fed through a dropout layer with dropout_rate equals to 0.5. It used the RELU activation function. The final dense layer was also fed through a dropout layer with dropout_rate equals to 0.5. It used the sigmoid activation function to drew out the predictions. To calculate the loss between the actual and predicted values, Binary Cross Entropy was used. This was minimized using the Adam optimizer in the training process. Keras Tuner was used for hyperparameter tuning which picked the optimal set of hyperparameters for the learning rate and the units of dense layer. These parameters were then used to initialize the final model and calculate the test set. Early Stopping was used to stop training early after reaching a certain value for the validation loss. Hyperparameters used in model training:

learning_rate = 0.01, dropout_rate = 0.5, batch_size = 32, epochs =11
From the image below, it can be seen that the the LSTM model, the data is fitting just right, it is neither over fitted nor under fitted, since both the validation and training curve are close by each other.



CNN: The third advanced model that we have implemented is CNN or the Convolutional Neural Network. The model is a deep learning model which is majorly used to classify images, but can also be used for sentiment analysis as used by us. Just like the above models, even here we have firstly used textVectorizer and then fed it to the embedding layer. After this we have used the Conv1D layer, with a kernel size of 8 and the relu activation function is used. After this layer MaxPooling1D with the pool size of 2 is done, now, in order to flatten the data, GlobalMaxPooling function is used. Finally, we have used two dense layers with relu, and sigmoid functions relatively. Also, both of these dense layers were fed through a dropout layer, with a dropout of 0.5. Rest of the part is similar to the above models.
For the CNN, both validation and training, loss curve and accuracy curve show amazingly fitted data, since the curves are smooth as well as very close to each other. Also the accuracy is also pretty good.



4.3. Error Analysis

Error analysis would help to us to diagnose the erroneous ML predictions which would give us a better understanding of the low and high performance of the models.Sentiment analysis are much more sensitive to stop words removal. Sometimes, it may happen that the overall meaning of the resulting sentence is positive after removing the stop words, which is not at all the reality. Punctuation is also very important to sentiments of the tweets as they it tells us the emotions. If the punctuation marks are removed, then the processed tweet meaning may change. So, it is necessary to check what the actual tweet looks like. When we analysed the samples that were misclassified by our model, we came across some examples where the meaning of the preprocessed tweet and the actual tweet was quite different due to the reasons discussed above, leading our model to make error while classifying the tweets. Also, there were some samples which were mislabelled in our dataset which might also be a reason behind the low accuracy and errors made by the model.

5. Results and Analysis

By using the Logistic regression as a baseline model, we obtained an accuracy of 91%. The advanced learning techniques included RNN, LSTM and CNN. These models gave an accuracy of 91%, 92% and 91% respectively. Which is still lower as compared to the ones in the papers above. Now, when the dataset is also large enough, the

reason for a lower accuracy could be the presence of wrongly classified data entries in the dataset. But still, we can conclude one thing, and that is, advanced learning techniques (specially LSTM) are giving higher accuracy as compared to the baseline model (Logistic Regression). Here, our limitation was the in availability of a proper and correctly classified dataset, due to which we couldn't achieve higher accuracy.

6. Contributions

6.1. Deliverables

1. **Rishita** : Preprocessing, data cleaning and feature extraction, RNN model
2. **Palak** : Training the baseline model, and building the CNN model
3. **Ishita**: Evaluation metrics and building the LSTM model.

All the deliverables mentioned above were delivered by each of the member.

6.2. Individual Contributions

1. **Rishita** : Preprocessing, data cleaning and feature extraction, RNN model, Analysis, Presentation, Report
Functions: The preprocessing and data cleaning function, feature extraction, Simple RNN, Classification Report, AUC ROC curve.
2. **Palak** : Training the baseline model, and building the CNN model, Analysis, Presentation, Report
Functions: Logistic Regression, CNN model, Classification Report, AUC ROC curve.
3. **Ishita**: Evaluation metrics and building the LSTM model, Analysis, Presentation, Report
Functions: LSTM model, Evaluation Metrics: Classification Report, AUC ROC curve, Confusion Metrics.

References

[1] Sentiment140 dataset with 1.6 million tweets. 2017. https://www.kaggle.com/kazanova/sentiment140, Sentiment analysis with tweets. 1

[2] Aileen Wang Diveesh Singh. Detecting depression through tweets. 6:1–9, 2018. 1

[3] Kuhaneswaran AL Govindasamy and Naveen Palanichamy. Depression detection using machine learning techniques on twitter data. In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 960–966, 2021. 1

[4] Md Islam, Muhammad Ashad Kabir, Ashir Ahmed, Abu Raihan M Kamal, Hua Wang, Anwaar Ulhaq, et al. Depression detection from social network data using machine learning techniques. *Health information science and systems*, 6(1):1–12, 2018. 1

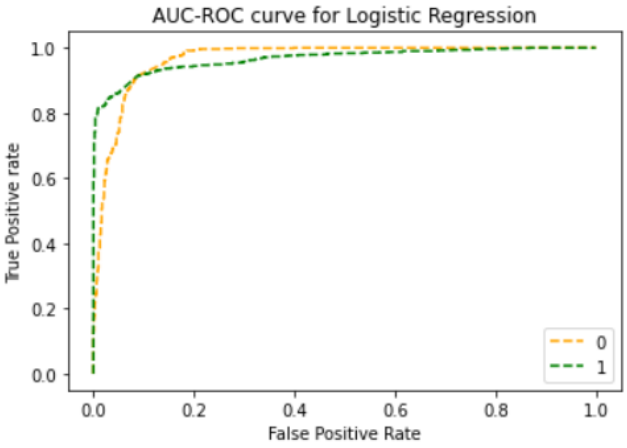


Figure 1. AUC - ROC curve for Logistic Regression

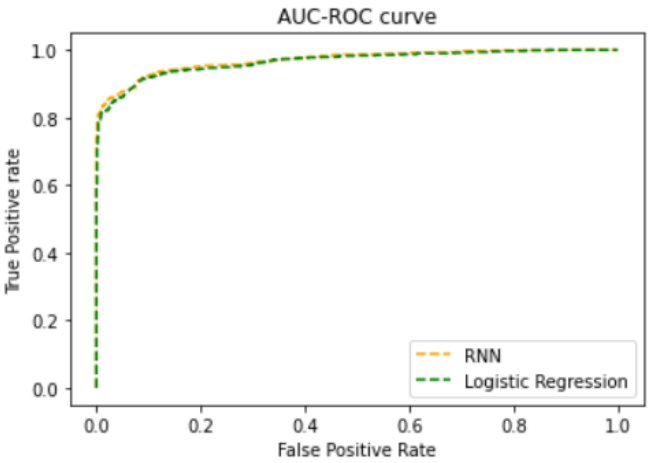


Figure 2. AUC - ROC curve for RNN

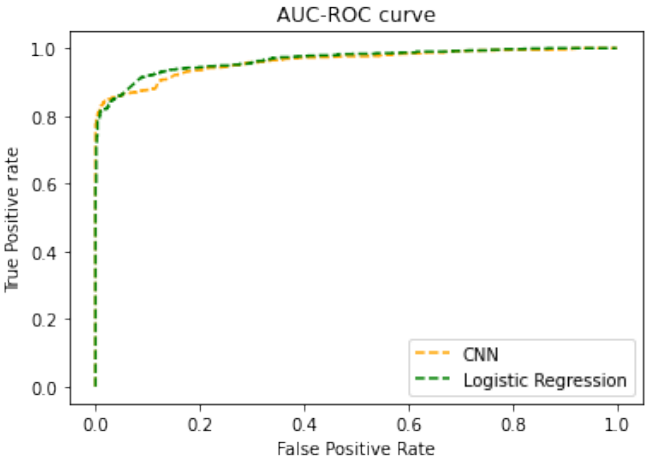


Figure 3. AUC - ROC curve for CNN

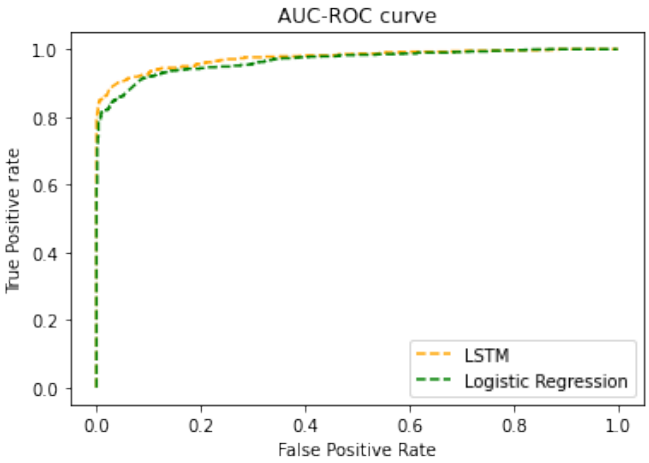


Figure 4. AUC - ROC curve for LSTM

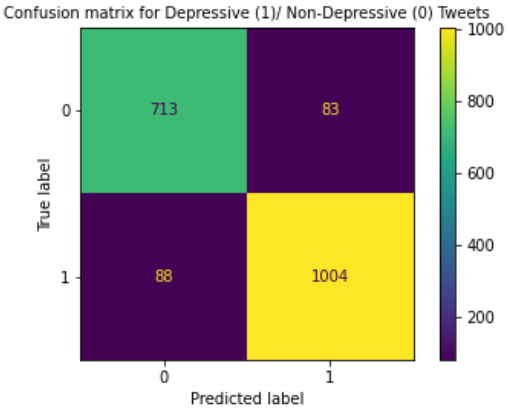


Figure 5. Confusion Matrix for Logistic Regression

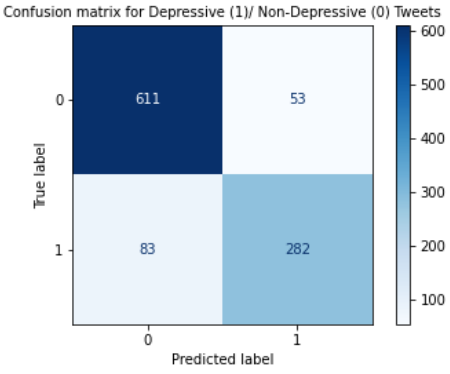


Figure 6. Confusion Matrix for RNN

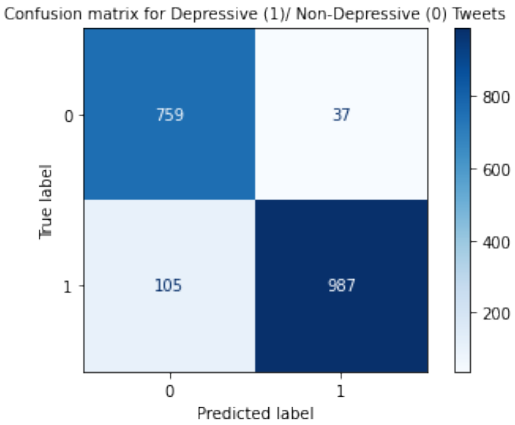


Figure 7. Confusion Matrix for LSTM

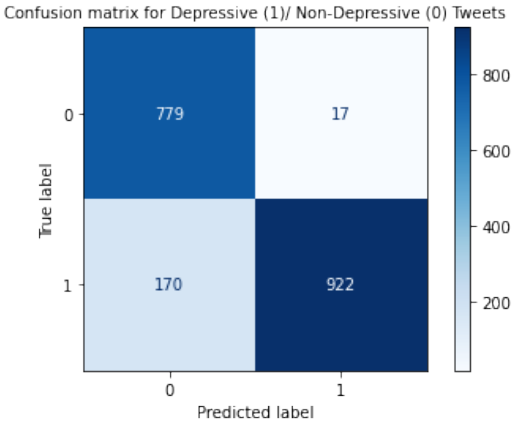


Figure 8. Confusion Matrix for CNN

	Class	Support	Recall	F1 - Score	Precision
CNN	0	796	0.98	0.89	0.82
	1	1092	0.84	0.91	0.98

Overall Accuracy: 0.91

Figure 9. Table for CNN

	Class	Support	Recall	F1 - Score	Precision
RNN	0	796	0.89	0.90	0.90
	1	1092	0.93	0.92	0.92

Overall Accuracy: 0.91

Figure 10. Table for RNN

	Class	Support	Recall	F1 - Score	Precision
Logistic Regression	0	796	0.90	0.89	0.89
	1	1092	0.92	0.92	0.92

Overall Accuracy: 0.91

Figure 11. Table for Logistic Regression

	Class	Support	Recall	F1 - Score	Precision
LSTM	0	796	0.95	0.91	0.88
	1	1092	0.90	0.93	0.96

Overall Accuracy: 0.92

Figure 12. Table for LSTM