Contents

Abstract

The main objective of our project is to conduct a comprehensive sentiment analysis on social media for 3200 tweets by calling Twitter's API using Python and classify expressed sentiments into positive, negative, or neutral categories. This analysis is critically important as it allows for a deeper understanding of public sentiment on social media platforms, offering crucial insights that can benefit businesses, policymakers, and social scientists. These insights are invaluable in developing more effective marketing strategies, informing public policy decisions, and gaining a delicate understanding of public attitudes towards a range of topics.

Our project involves using advanced Natural Language Processing techniques to precisely decode and assess the sentiments embedded within textual data. By employing methods such as machine learning algorithms, and deep learning models like LSTM or Transformer-based networks, our project aims to enhance the accuracy and reliability of sentiment detection. Furthermore, our project will explore the use of various data preprocessing and feature extraction strategies to better handle the complexities and nuances of language used in social media, such as emojis, and abbreviations, which are often challenging for standard NLP tools.

1. Data Collection & Cleansing

We downloaded the Tweets data (sentiment140 dataset) from the Kaggle website [1]. The dataset contains 1,600,000 tweets extracted using the twitter API. The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment. It has 6 fields: target (the sentiment of the tweet, 0 = negative, 4 = positive), ids (the ID of the tweet), date, flag (the query), user, text. Among these fields, we only kept the text and target (renamed as "label", 0 = negative, 1 = positive) for our study. From the large dataset, we randomly selected 3200 samples for our model training and test.

As the text contains many non-alphabet characters and special contents irrelevant to the sentiment analysis, we conducted the data cleansing and removed the noises. In the data cleansing stage, we designed a function TwitterCleaning() to remove all the mentions, numbers, URLs and special characters in the text. Besides, the samples with text length less than 3 are deleted from the dataset to ensure the text is meaningful. We have 3190 samples left after data cleansing.

We examined the balance of the dataset by calculating the proportion of the number of samples labeled as 0 and the number of samples labeled as 1. It turned out that the positive text accounts for 50.34% and the negative text accounts for 49.66% in this dataset, which indicated the dataset is perfectly balanced.

2. Data Exploratory Analysis

2.1 WordCloud

We used the WordCloud to explore the frequent words appearing in the positive Tweets and negative Tweets. Common English stopwords enhanced with some custom stopwords (includes "Im", "day", "today", "going", "go", "get") are applied to eliminate frequent but irrelevant words. The word clouds were generated using Python's WordCloud library, with separate visualizations for each sentiment category.

Figure 1.Positive & Negative WordClouds

From the above graph, we can see that words like “good”, “love”, “thank” are frequently emerged in the positive texts while words like “work”, “cant”, miss”, “sorry”, “sick” are frequently emerged in the negative texts. And the word “like” is the frequent word in both positive and negative texts.

2.2 Word List

We use the function ‘wordList’ to analyze text data from a dataframe and identify frequent words associated with different labels ('positive' and 'negative' sentiments). Frequent words in positive sentences are usually ['in', 'i', 'are', 'day', 'your', 'good', 'love', 'from', 'do'] and frequent words in negative sentences are usually ['for', 'you', 'not', 'go', 'work', 'no', 'dont', 'cant', 'miss'].

2.3 Distribution of the length of sentences

We also investigated the distribution of the length of sentences in the dataset.

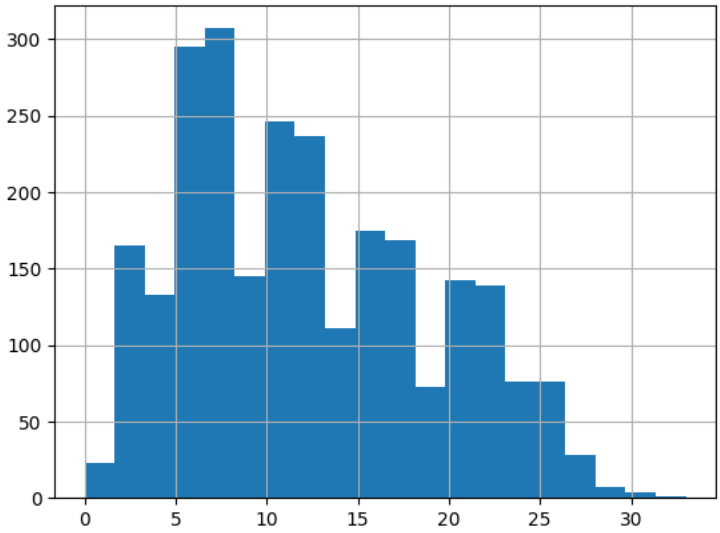

Figure 2. Distribution of the length of sentences

The maximum length of sentences in the dataset is 143 and the mean length of sentences in the dataset is around 63.

3. Data Preprocessing

3.1 Train & Validation set split

The primary reason for dividing the dataset into training and validation sets is to ensure that our model can be trained on one subset of the data (training set) and then evaluated on

a separate, unseen subset (validation set). This helps in assessing our model's performance and generalization ability.

We use ‘train_test_split’ to randomly divide the dataset into training and validation sets. After that we set the parameter test_size=0.2 which specifies that 20% of the data should be set aside for the validation set, while the remaining 80% will be used for training. The entire dataset is split into the training set (80%) and the validation set (20%). The training set contains 2552 samples and the validation set contains 638 samples. The split is stratified by the labels to ensure the samples are balanced in both the training set and the validation set.

3.2 Sentence Embedding

We used TfidfVectorizer to transform the text string into digital format. TfidfVectorizer is a feature extraction tool from the sklearn.feature_extraction.text module of the Scikit-learn library, commonly used for text analysis tasks in machine learning. It converts a collection of raw documents into a matrix of TF-IDF features. TF-IDF stands for Term Frequency-Inverse Document Frequency. It is a statistical measure used to evaluate the importance of a word in a document in a collection or corpus of documents. The importance increases proportionally with the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

4. Classification Models

4.1 Three classifiers: Logistic Regression, SVM, Random Forest

In our first classification model, we choose Logistic Regression, Support Vector Machine (SVM), and Random Forest to build the classification model. These are three popular machine learning algorithms used for classification tasks. Each has distinct characteristics and applications depending on the nature of the problem and data.

Logistic Regression Classifier: In NLP and sentiment analysis, Logistic Regression is particularly useful for binary classification tasks, such as determining whether a given text expresses a positive or negative sentiment. Logistic Regression is easy to implement and provides a robust baseline for any binary classification problem, also it is computationally less intensive, making it a fast model for training on large datasets.

SVM Classifier: In sentiment analysis, SVMs are appreciated for their effectiveness in handling high-dimensional data, such as text data where each word can be treated as a feature. The kernel trick is an essential feature of SVM, allowing it to solve complex problems. This makes SVMs highly adaptable to various types of data.

Random Forest Classifier: Random Forest is an ensemble learning method for classification that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. It is

particularly well-suited for handling the non-linear relationships that are often present in text data.

For the 5-fold cross validation, each model undergone grid search cross-validation to find the best set of parameters, and the best model is then evaluated on a validation set. The variable "param_grids" specifies the hyperparameters to tune for each model through cross-validation:

- Logistic Regression: Regularization parameter (C) with values ranging from 0.01 to 100
- Support Vector Machine: Regularization parameter (C) with values ranging from 0.1 to 10 and the kernel type (linear or rbf)
- Random Forest: Number of trees (n_estimators) with values ranging from 100 to 300 and the number of features to consider when looking for the best split (max_features)

4.2 Ensemble methods with voting classifier

We tried to ensemble the above three classifiers and get the final result by a voting classifier. In this model, we added a Gradient Boosting Classifier, which is believed to improve the overall performance. The VotingClassifier is set up with voting='soft', which means it predicts the class label based on the argmax of the sums of the predicted probabilities. This method can provide more weight to highly confident votes and is generally more flexible than hard voting, which only considers predicted labels.

4.3 Deep Learning models

4.3.1 Neural Networks

Our Neural Networks has the following architecture:

- Input Layer: Takes input with a dimensionality equal to the number of features in the TF-IDF vectorized data.
- Dense Layers:
  First Dense Layer: 512 neurons with ReLU (Rectified Linear Unit) activation. ReLU is used for its efficiency and ability to provide non-linear transformation.
  Second Dense Layer: 256 neurons, also with ReLU activation. This layer hierarchy increases the network's ability to learn complex patterns in the data.
- Dropout Layers: Each with a dropout rate of 0.5. These layers randomly set a fraction of input units to 0 at each update during training time, which helps prevent overfitting.
- Output Layer: A single neuron with a sigmoid activation function, making it suitable for binary classification. The sigmoid function outputs probabilities that are interpreted as the likelihood of the input being in the positive class.

4.3.2 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a language model based on the transformer architecture, which is commonly applied in Natural Language Processing and is demonstrated to be powerful in sentiment analysis tasks.

For this task, we adopted the 'bert-base-uncased' pretrained BERT model and utilized the BertTokenizerFast for tokenizing input text, ensuring compatibility with the BERT model by matching tokenization with BERT's training.

Model architecture:

- Encoder: Incorporates the BERT model as a fixed feature extractor, with an option to fine-tune (train_encoder).
- LSTM Layer: Processes the sequence of embeddings output by BERT, capturing temporal dependencies among encoded tokens.
- Dropout Layers: Applied post-LSTM and post-dense layer to reduce overfitting by randomly setting input units to 0 at a rate of 0.1.
- Dense Layers: Two dense layers with ReLU and softmax activations, respectively, with the final layer outputting probabilities across two classes.

5. Results

We used the following metrics to evaluate the model performance: accuracy, F1-score, precision, recall, AUC score. The ROC curve is plotted in each model to show the performance at all classification thresholds.

5.1 Three classifiers: Logistic Regression, Support Vector Machine, Random Forest

(i) Logistic Regression

Cross validation result: The best parameters for Logistic Regression is: Regularization parameter C = 1.

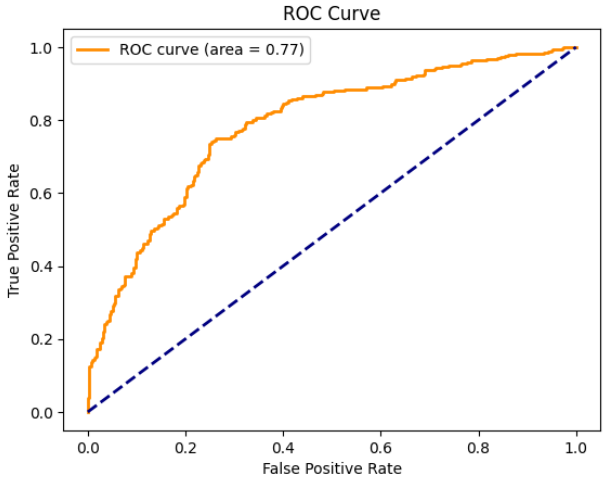

Figure 3. ROC curve of Logistic Regression

Table 1: Results for LR

Metrics	LR
Accuracy	0.74
F1-score	0.76
Precision	0.73
Recall	0.76
ROC AUC	0.7850

(ii) Support Vector Machine

Cross validation result: The best parameters for Logistic Regression is: Regularization parameter C = 1 and kernel = 'rbf'

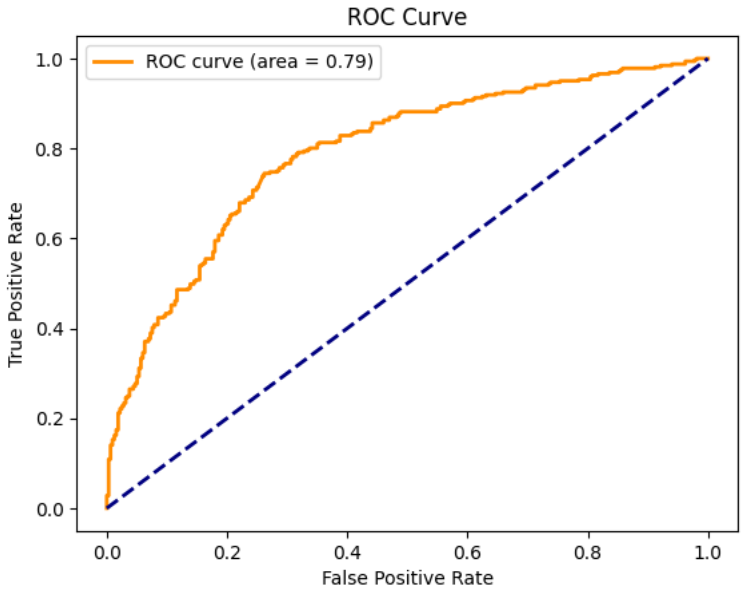

Figure 4. ROC curve of Support Vector Machine

Table 2: Results for SVM

Metrics	SVM
Accuracy	0.73
F1-score	0.73
Precision	0.73
Recall	0.73
ROC AUC	0.7884

(iii) Random Forest

Cross validation result: The best parameters for Logistic Regression is: max_features = 'log2' and n_estimators = 200.

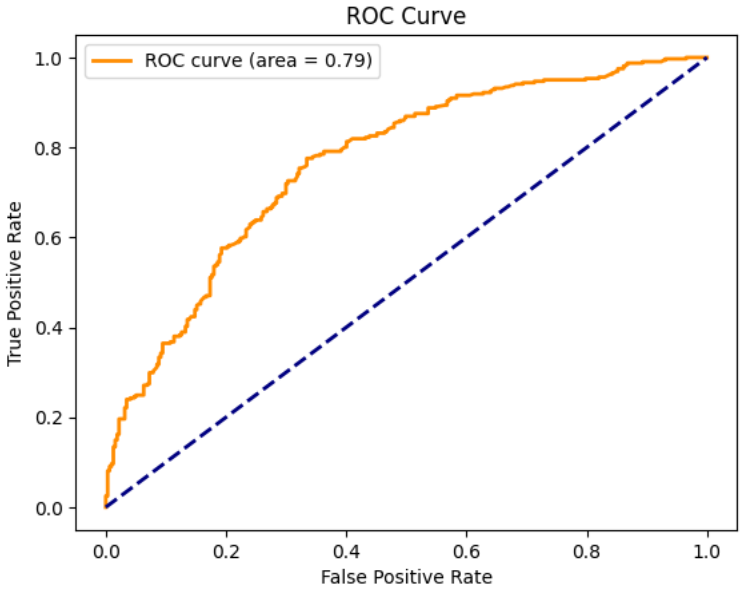

Figure 5. ROC curve of Random Forest

Table 3: Results for RF

Metrics	RF
Accuracy	0.70
F1-score	0.71
Precision	0.70
Recall	0.72
ROC AUC	0.7623

Comparing these three models, Logistic Regression and SVM are particularly strong when having high-dimensional sparse data, as is typical in text classification. They offer good performance and are easy to implement, with SVM providing additional flexibility through the kernel trick. Random Forest, being an ensemble method, offers robustness and performs well on complex datasets where relationships between features are non-linear.

From the above three plots we can find that the AUC for Logistic Regression is 0.7858, the AUC for SVM is 0.7898, the AUC for Random Forest is 0.7657. The AUC for SVM is the biggest, which means that the SVM model is the best. Also, as for the recall and f1-score,the recall and f1-score for Logistic Regression is 0.73, the recall and f1-score for SVM is 0.74, the recall and f1-score for Random Forest is 0.71, which means that the SVM model is the best, and the Logistic Regression is the second best. In contrast, Random Forest showed the weakest performance among the models tested.

5.2 Voting classifier

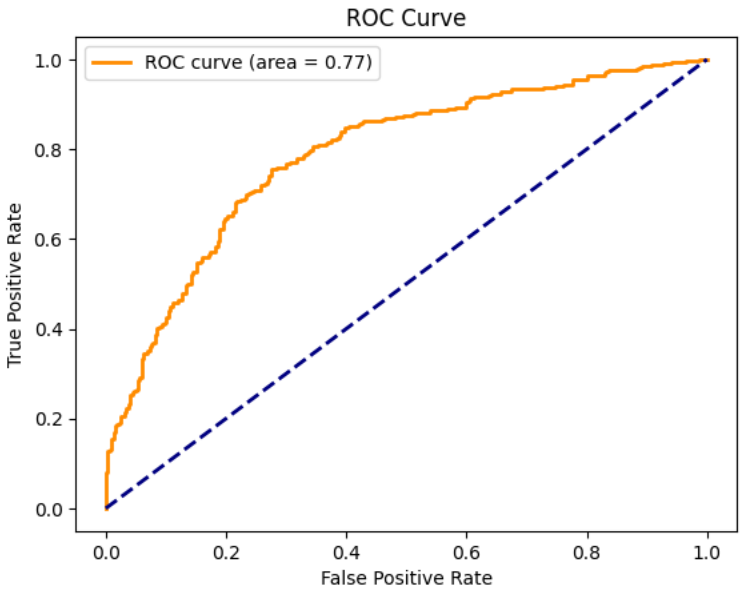

Figure 6. ROC curve of Voting Classifier

Table 4: Results for Voting Classifier

Metrics	Voting
Accuracy	0.73
F1-score	0.73
Precision	0.74
Recall	0.72
ROC AUC	0.7893

5.3 Neural Networks

Table 5: Results for Neural Networks

Metrics	NN
Accuracy	0.68
F1-score	0.67
Precision	0.69
Recall	0.66
ROC AUC	0.7400

5.4 BERT

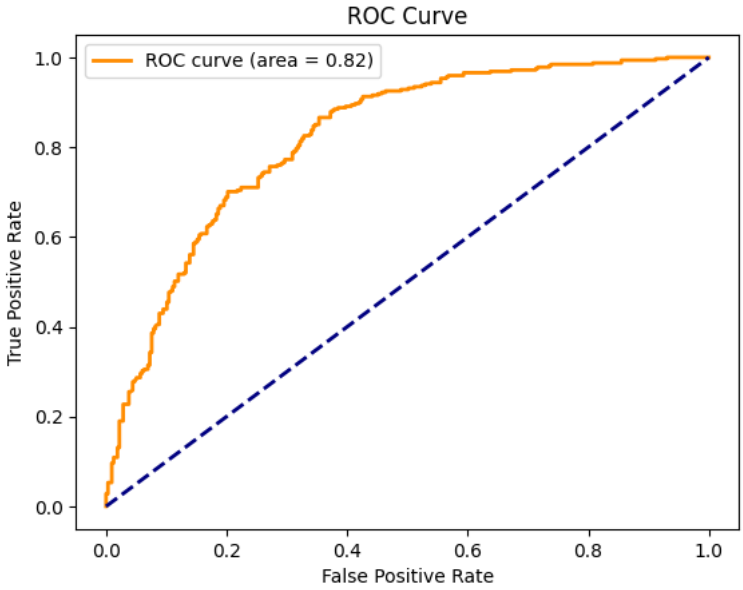

Figure 7.ROC curve of BERT

Table 5: Results for BERT

Metrics	BERT
Accuracy	0.74
F1-score	0.75
Precision	0.73
Recall	0.78
ROC AUC	0.8058

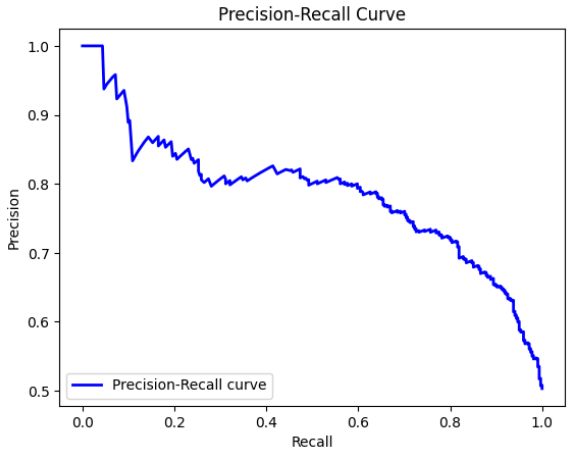

Figure 8.Precision-Recall curve for BERT

The above graph is a Precision-Recall Curve. Precision (y-axis) measures the accuracy of positive predictions made by the classifier, formulated as the ratio of true positives (TP) to the sum of true positives and false positives (TP + FP). Recall (x-axis), measures the ability of the classifier to find all the positive samples. It is calculated as the ratio of true positives (TP) to the sum of true positives and false negatives (TP + FN). From our graph, it appears that: At lower recall levels, the precision starts very high (close to 1.0) but quickly drops, indicating that when attempting to cover only a small portion of the actual positives (low recall), the classifier can be very precise. As the recall increases, the precision generally decreases, which is typical as the classifier begins to include more positive predictions, some of which are incorrect (false positives increase).

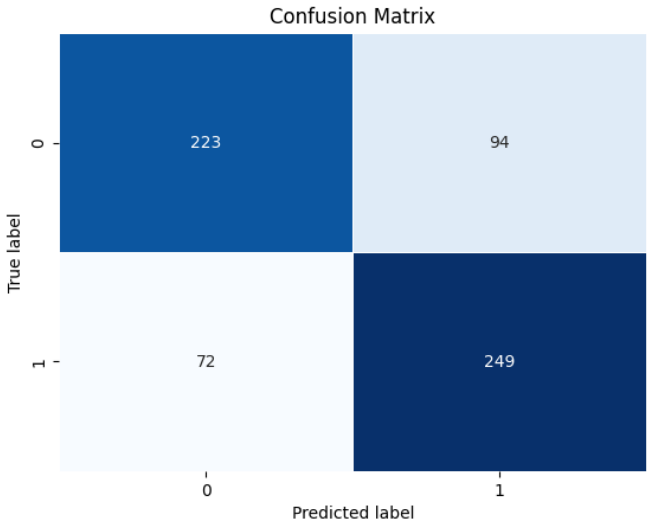

Figure 9.Confusion matrix for BERT

The top left cell (223) represents the number of true positives (TP), where the actual class is 0 and the model correctly predicted 0. The top right cell (94) represents the number of false negatives (FN), where the actual class is 0 but the model incorrectly predicted 1. The bottom left cell (72) represents the number of false positives (FP), where the actual class is 1 but the model incorrectly predicted 0. The bottom right cell (249) represents the number of true negatives (TN), where the actual class is 1 and the model correctly predicted 1.

6. Discussion

From the table, the BERT model shows the best overall performance for the given dataset in our six models, with the accuracy achieving 0.74 and AUC score 0.8058. Among the three machine learning models (Logistic Regression, Support Vector Machine and Random Forest), the Logistic Regression model and the SVM model performs slightly better than the Random Forest on this dataset. The voting classifier, which considers all the results of the previous three classifiers, gives similar results with the best models among the three

classifiers.

Table 1: Results for different models

Metrics	LR	SVM	RF	Voting	NN	BERT
Accuracy	0.74	0.73	0.70	0.73	0.68	0.74
F1-score	0.76	0.73	0.71	0.73	0.67	0.75
Precision	0.73	0.73	0.70	0.74	0.69	0.73
Recall	0.76	0.73	0.72	0.72	0.66	0.78
ROC AUC	0.7850	0.7884	0.7623	0.7893	0.7400	0.8058

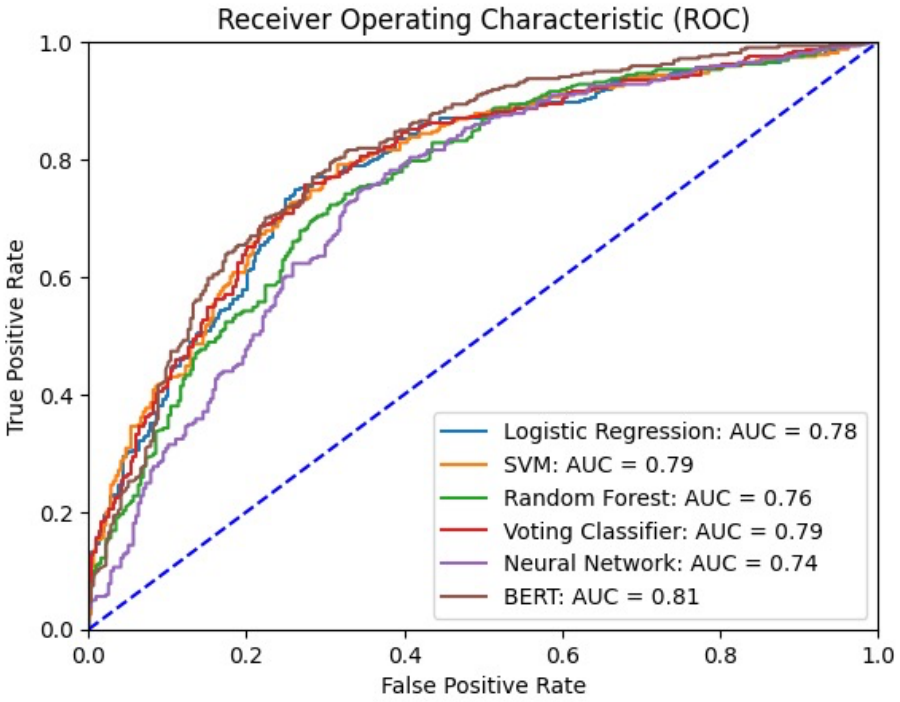

Figure 10.ROC curve for different models

References

[1] Sentiment140 dataset with 1.6 million tweets:
https://www.kaggle.com/datasets/kazanova/sentiment140/data