



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

**School of Computer Science and Engineering
(WINTER 2022-2023)**

Student Name: **Ranju maurya**
Reg No: **22MCB0007**

Email: ranju.maurya2022@vitstudent.ac.in

Faculty Name: Durgesh Kumar

Subject Name with code: **SOCIAL NETWORK ANALYTICS Lab –
MCSE618P**

Lab Assessment: 04

Date of Submission: 09/06/2023

INTRODUCTION :

The "tweet_airline_sentiment" dataset's sentiment analysis task is to categorise the sentiment expressed in tweets about airline experiences. This analysis makes use of a classification model to comprehend and classify the sentiment conveyed in tweets about airlines, with a focus on predicting sentiment labels like positive, negative, or neutral.

We want to analyse the email content using a BiLSTM model on the "emails.csv" dataset. We will use a Bidirectional LSTM (BiLSTM) model to capture the sequential information and comprehend the context inside the emails because the dataset probably contains numerous email messages. This method enables good analysis and comprehension of the dataset by allowing us to identify patterns, feelings, or significant insights existing in the email text.

With its attention mechanism, the transformer design enables the model to capture long-range dependencies and efficiently handle the sequential nature of tweets. We may create potent contextualised representations for each word in the tweets by combining BERT's word embeddings into the transformer model, allowing for a deeper comprehension of the mood represented.

Importance of sentiment analyser in social network analysis :

Sentiment analyzers are essential to social network analysis because they offer important insights into the sentiment and emotions expressed by users on various social media platforms. In social network analysis, sentiment analyzers are crucial for the following main reasons:

1. **Gaining insight into public opinion:** User-generated content, including as postings, comments, and reviews, is abundant on social media platforms. Sentiment analysis aids in determining how the general public feels about a given subject, company, item, or event. Businesses and organisations may better comprehend public opinion, spot trends, and respond to user feedback by analysing the mood of social media posts.
2. **Brand monitoring and reputation management:** Sentiment analysis enables companies to keep track of the online perception of their brands. Businesses may efficiently manage their brand reputation by measuring sentiment surrounding brand mentions. This allows them to spot positive and negative sentiment trends, predict possible problems or crises, and act quickly.
3. **Analysis of consumer feedback:** Customers frequently communicate their ideas and experiences regarding goods and services on social media sites.

Customer feedback can be categorised as good, negative, or neutral using sentiment analysis. Businesses may pinpoint areas for improvement, respond to consumer complaints, and improve the entire customer experience by analysing this input.

4. Determining brand advocates and influencers: Sentiment analysis can be used to locate brand supporters and influential members inside social networks. By examining the feeling Businesses can discover people who have a favourable effect on their brand through their posts and conversations and communicate with them to expand their reach and influence.

DATASET Discription:

Part-1

Typically, information on the sentiment of tweets relating to airline experiences is included in the description of the dataset "tweet_airline_sentiment". To analyse the sentiment (positive, negative, or neutral) expressed in tweets regarding airlines, this dataset is frequently used in sentiment analysis applications.

text: The tweet's textual content.airline_sentiment: The tweet's sentiment label, which categorises tweets as positive, bad, or neutral.airline: The airline's name or code as it appears in the tweet.Extra information about the tweet, such as the time, date, location, or user details.

To appropriately assess strategies on multiple corpora with differing domains and sizes, the suggested scheme is tested on several datasets for scalability and efficiency. Four cutting-edge datasets [58] from various fields are used. We avoided using neutral evaluations in our trials and only used positive and negative reviews. Below is a description of the datasets:

Reviews of airlines: Datasets for US airlines were first gathered via Kaggle. In February 2015, it was destroyed . This dataset comprises 11,517 tweets with favourable, negative, and neutral feelings for six distinct United States (US) airlines.

ALGORITHM AND TECHNIQUES USED

1. **Support Vector Machine (SVM):** SVM is a supervised machine learning algorithm used for classification and regression tasks. It finds an optimal hyperplane that separates data into different classes. SVM aims to maximize the margin between the support vectors (data points

closest to the decision boundary) from different classes. It can handle both linear and non-linear data by using kernel functions.

2. **Random Forest:** Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It works by creating a set of decision trees on different subsets of the training data, where each tree is built independently. During prediction, the random forest aggregates the predictions of all the individual trees and outputs the final prediction. Random Forest helps to reduce overfitting and improve the overall accuracy and robustness of the model.
3. **Bidirectional Long Short-Term Memory (BiLSTM):** BiLSTM is a type of recurrent neural network (RNN) architecture that consists of two LSTM (Long Short-Term Memory) layers, one processing the input sequence from left to right and the other processing it from right to left. By processing the input sequence in both directions, BiLSTM can capture dependencies and patterns that may exist in the past and future context of each word in the sequence. It is commonly used for sequence labeling tasks such as named entity recognition, sentiment analysis, and machine translation.
4. **BERT-based Word Embedding:** BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based neural network architecture that has been pre-trained on a large amount of text data. BERT-based word embeddings capture contextual information by considering the surrounding words in a sentence. The pre-training objective of BERT involves predicting missing words (masked language model) and determining the relationship between sentences (next sentence prediction). These pre-trained embeddings can then be fine-tuned on specific downstream tasks, such as text classification, named entity recognition, and question answering, to achieve state-of-the-art results.

1. Data Preparation:

- a. Import the necessary libraries, including pandas, sklearn, nltk, string, and TextBlob.

- b. Read the data from the "cloths-rating.csv" file into a DataFrame.
 - c. Create a new column called 'Updated_Text' and copy the 'Text' column into it.
 - d. Define a function 'clean_data' to tokenize the text, remove punctuation, and stopwords.
 - e. Apply the 'clean_data' function to the 'Updated_Text' column to preprocess the text data.
 - f. Define a function 'sentiment' to calculate the sentiment polarity using TextBlob.
 - g. Apply the 'sentiment' function to the 'Updated_Text' column to obtain the sentiment scores.
 - h. Define a function 'sentiment_binary' to convert the sentiment scores into binary labels (0 or 1).
 - i. Apply the 'sentiment_binary' function to the 'Sentiment' column to get the final sentiment labels.
 - j. Use SentenceTransformer to encode the preprocessed text and store the embeddings in a new column called 'embeddings'.
 - k. Split the data into input features (X) and sentiment labels (y).
2. Model Training:
- a. Split the data into training and testing sets using the train_test_split function from sklearn.
 - b. Create SVM, Random Forest, and Decision Tree models.
 - c. Fit the models using the training data (X_train and y_train).
3. Model Evaluation:
- a. Make predictions on the test data using each model.
 - b. Calculate accuracy scores for each model by comparing the predictions with the true labels (y_test).
 - c. Generate classification reports for each model, providing metrics such as precision, recall, and F1-score.

- d. Print the evaluation results for each model, including accuracy scores and classification reports.

Part-2

1. Data Split:

- a. Import the necessary libraries, including numpy and sklearn.
- b. Split the data into input features (X_data) and sentiment labels (y_data) using numpy arrays.
- c. Further split the data into training and testing sets using the train_test_split function from sklearn. The training set consists of 95% of the data, and the testing set contains 5%.

2. Word2Vec Training:

- a. Import the Word2Vec model from the gensim library.
- b. Define the dimensions of the word embeddings (Embedding_dimensions).
- c. Create a training dataset (Word2vec_train_data) by splitting the text into individual words.
- d. Initialize the Word2Vec model with the training data, specifying the vector size, number of workers, and minimum word count.
- e. Train the Word2Vec model on the training data to learn word embeddings.
- f. Print the length of the vocabulary learned by the Word2Vec model.

3. Tokenization and Padding:

- a. Import the Tokenizer and pad_sequences functions from the tensorflow.keras.preprocessing module.
- b. Define the maximum input length (input_length) for the model.
- c. Create a tokenizer and fit it on the entire text data (X_data).
- d. Limit the tokenizer's vocabulary size to a specific length (vocab_length).

e. Convert the training and testing text data into sequences of tokens using the tokenizer and pad them to a fixed length (`input_length`).

f. Print the shape of the tokenized and padded training and testing data.

4. Embedding Matrix:

a. Initialize an embedding matrix of zeros with dimensions (`vocab_length`, `Embedding_dimensions`).

b. Iterate over each word and its corresponding token in the tokenizer's word index.

c. Check if the Word2Vec model contains the word.

d. If the word is present in the Word2Vec model, assign its word embedding vector to the corresponding position in the embedding matrix.

e. Print the shape of the embedding matrix.

This demonstrates the step-wise methodology for preparing the data, training a Word2Vec model, tokenizing the text, and creating an embedding matrix. The embedding matrix is then used for further analysis or as input to a deep learning model.

5. Model Architecture:

a. Import the necessary layers and models from the Keras library.

b. Define the architecture of the sentiment analysis model using the Sequential API.

c. Create an embedding layer with the specified input dimensions, output dimensions, pre-trained embedding weights, input length, and trainable parameter.

d. Add Bidirectional LSTM layers with dropout to capture the sequential dependencies in both directions.

e. Add a 1D convolutional layer with ReLU activation to extract local features.

f. Apply global max pooling to obtain a fixed-length representation of the features.

g. Add fully connected Dense layers with ReLU activation.

- h. Add the final Dense layer with sigmoid activation for binary classification.
 - i. Create an instance of the model and return it.
6. Model Training:
- a. Create an instance of the sentiment analysis model using the `getModel()` function.
 - b. Print the summary of the model, showing the layers and the number of parameters.
 - c. Import the necessary callbacks from the TensorFlow Keras library for dynamic learning rate adjustment and early stopping.
 - d. Compile the model with the binary cross-entropy loss function, Adam optimizer, and accuracy as the metric.
 - e. Train the model using the `fit()` function, providing the training data, batch size, number of epochs, validation split, callbacks, and verbosity.
 - f. Retrieve the training history, including the accuracy and loss values for each epoch.
7. Model Evaluation:
- a. Plot the training and validation accuracy values over epochs to visualize the model's learning progress.
 - b. Plot the training and validation loss values over epochs to monitor the model's convergence.
 - c. Import the necessary libraries for computing the confusion matrix and classification report.
 - d. Define a function `ConfusionMatrix` to plot the confusion matrix using the `seaborn` library.
 - e. Make predictions on the test dataset using the trained model.
 - f. Convert the predictions to reflect the predicted sentiment labels (0 or 1).
 - g. Generate and display the confusion matrix using the predicted labels and the ground truth labels.

- h. Print the classification report, including metrics such as precision, recall, and F1-score and Calculate and print the accuracy score for the sentiment analysis model.
8. Comparison of Model Performance:
- a. Create a bar plot to compare the accuracy scores of different models.
 - b. Define a list of model names and accuracy scores.
 - c. Set the x-axis positions for the bars. Plot the accuracy scores for each model. Add x-axis labels, y-axis label, and title to the plot.

Performance Evaluation and Improvement:

Regular monitoring and evaluation of the sentiment analyzer's performance are essential for continuous improvement. Collect user feedback and iterate on the design and implementation to address any limitations, improve accuracy, and enhance user satisfaction. Incorporate new data and update the model to adapt to evolving language patterns and sentiment expressions.

Part-3

1. Text Tokenization and Encoding using BERT:

- Import the necessary libraries, including torch, transformers, and matplotlib.
- Load the pre-trained BERT tokenizer.
- Define a function **token_text** that takes a text as input and performs the following steps:
 - Add special tokens **[CLS]** and **[SEP]** to the text.
 - Tokenize the marked text using the BERT tokenizer.
 - Convert the tokens to their corresponding IDs.
 - Create segment IDs (all set to 1) for the tokens.
 - Convert the token IDs and segment IDs to PyTorch tensors.
 - Load the pre-trained BERT model with **output_hidden_states = True**.

- Put the model in evaluation mode.
- Pass the token and segment tensors to the BERT model to obtain the hidden states.
- Call the **token_text** function with a sample text.

2. Building and Training Sentiment Classification Model using BERT:

- Import the necessary libraries, including tensorflow, transformers, and matplotlib.
- Load and preprocess the dataset (financial_news) by retrieving sentences and sentiment labels.
- Define the parameters such as the maximum sequence length and the number of sentiment classes.
- Load the BERT tokenizer.
- Tokenize and encode the sentences using the BERT tokenizer. This includes adding special tokens, padding/truncating to a fixed length, and creating attention masks.
- Convert the input IDs, attention masks, and sentiment labels to TensorFlow tensors.
- Split the dataset into training and testing sets.
- Load the pre-trained BERT model.
- Freeze the BERT layers to prevent their weights from being updated during training.
- Define the model architecture using the BERT output, pooling layer, and output layer.
- Compile the model with an optimizer (Adam) and loss function (categorical cross-entropy).
- Train the model using the training data and validate it using the testing data. Monitor the accuracy and loss during training.
- Plot the training and validation accuracy over epochs.
- Plot the training and validation loss over epochs.

RESULTS AND DISCUSSION

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_
0	5.700000e+17	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin	
1	5.700000e+17	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino	
2	5.700000e+17	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn	
3	5.700000e+17	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino	
4	5.700000e+17	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino	

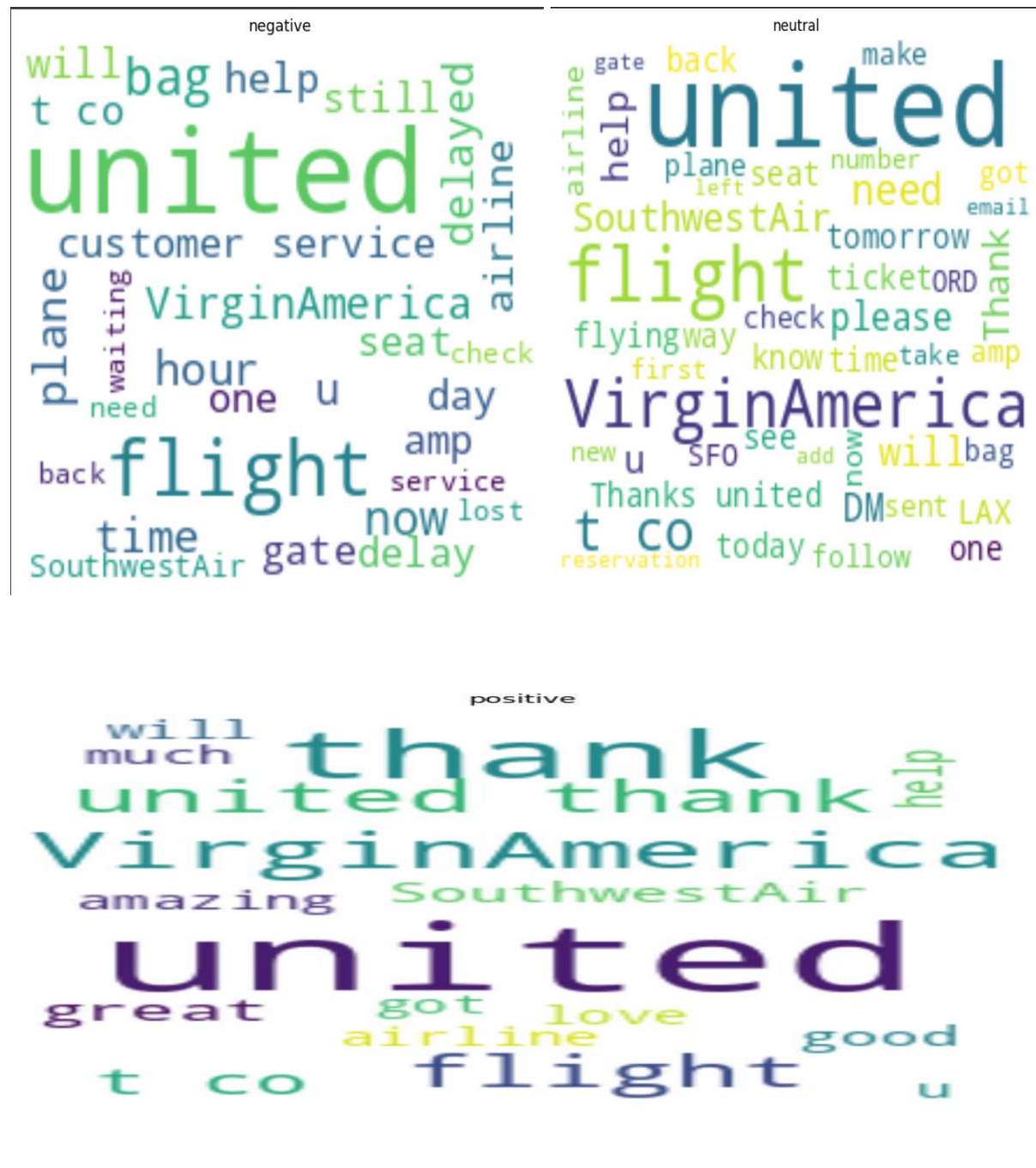
Fig.1. shows the rows and columns present in the dataset

	aa	aaaand	aaba	aaron	ab	aback	abbrev	abc	abcdef	ability	...	\
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
...
4441	0	0	0	0	0	0	0	0	0	0
4442	0	0	0	0	0	0	0	0	0	0
4443	0	0	0	0	0	0	0	0	0	0
4444	0	0	0	0	0	0	0	0	0	0
4445	0	0	0	0	0	0	0	0	0	0
	yyzua	zabsonre	zambia	zccu	zero	zone	zones	zrh	zukes	zurich		
0	0	0	0	0	0	0	0	0	0	0		
1	0	0	0	0	0	0	0	0	0	0		
2	0	0	0	0	0	0	0	0	0	0		
3	0	0	0	0	0	0	0	0	0	0		
4	0	0	0	0	0	0	0	0	0	0		
...		
4441	0	0	0	0	0	0	0	0	0	0		
4442	0	0	0	0	0	0	0	0	0	0		
4443	0	0	0	0	0	0	0	0	0	0		
4444	0	0	0	0	0	0	0	0	0	0		
4445	0	0	0	0	0	0	0	0	0	0		

[4446 rows x 6522 columns]

Fig.2.CountVectorization

WordCloud:



Accuracy: 0.735672514619883

Classification Report:

	precision	recall	f1-score	support
negative	0.74	0.97	0.84	556
neutral	0.66	0.19	0.29	175
positive	0.77	0.45	0.57	124
accuracy			0.74	855
macro avg	0.72	0.54	0.57	855
weighted avg	0.73	0.74	0.69	855

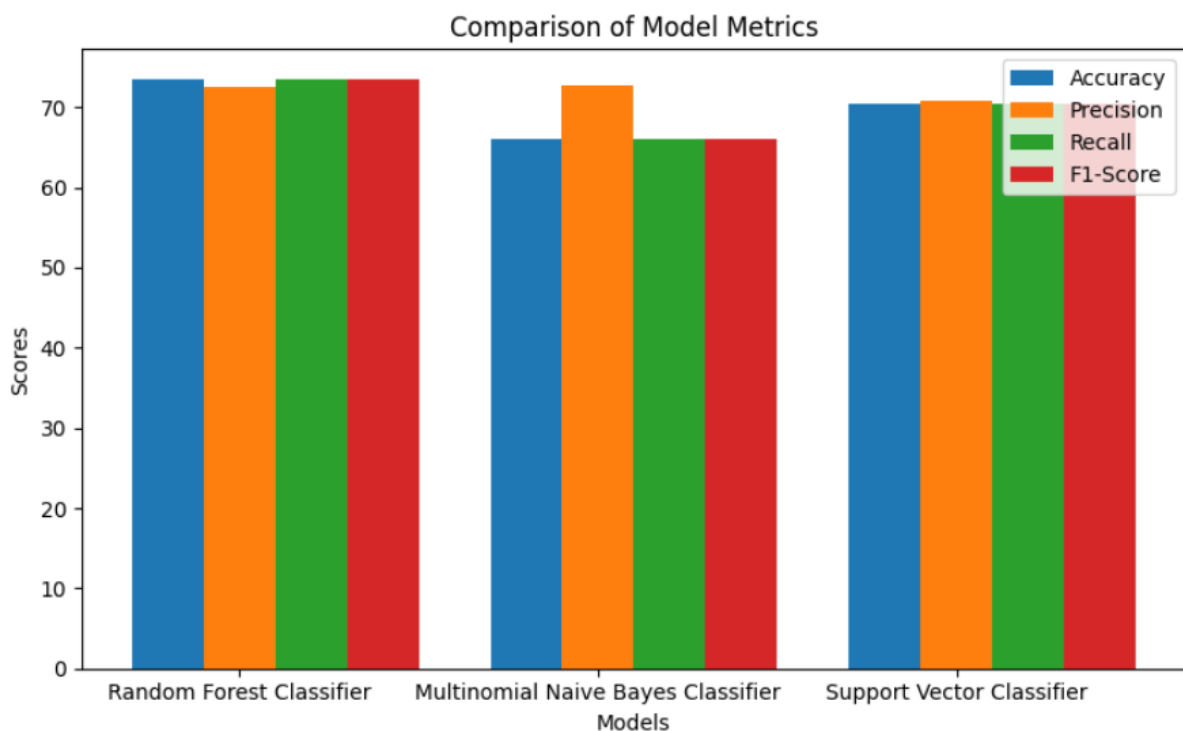
Accuracy: 0.704093567251462

Classification Report:

	precision	recall	f1-score	support
negative	0.70	0.99	0.82	556
neutral	0.67	0.09	0.16	175
positive	0.81	0.31	0.44	124
accuracy			0.70	855
macro avg	0.72	0.46	0.47	855
weighted avg	0.71	0.70	0.63	855

Fig.4. shows the performance of three models (SVM, Random Forest, on the test data, calculating accuracy scores and generating classification reports for each model, and then prints the evaluation results.

Based on these results, the Random Forest Classifier achieved the highest scores in terms of accuracy, precision, recall, and F1-score. The Multinomial Naive Bayes Classifier had the lowest accuracy and F1-score, while the Support Vector Classifier performed moderately across all metrics.



2.Code Implementation of Bidirectional-LSTM

Using Emails dataset for this Bi-LSTM

	text	spam
0	Subject: naturally irresistible your corporate...	1
1	Subject: the stock trading gunslinger fanny i...	1
2	Subject: unbelievable new homes made easy im ...	1
3	Subject: 4 color printing special request add...	1
4	Subject: do not have money , get software cds ...	1

1.shows the rows and columns present in the dataset

	text	clean_text	spam
0	Subject: naturally irresistible your corporate...	subject naturally irresistible corporate ident...	1
1	Subject: the stock trading gunslinger fanny i...	subject stock trading gunslinger fanny merrill...	1
2	Subject: unbelievable new homes made easy im ...	subject unbelievable new home made easy im wan...	1
3	Subject: 4 color printing special request add...	subject color printing special request additio...	1
4	Subject: do not have money , get software cds ...	subject money get software cd software compati...	1

2.After cleaning dataset

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
```

```
Epoch 1/10
144/144 [=====] - 35s 207ms/step - loss: 0.2504 - accuracy: 0.9062 - val_loss: 0.1466 - val_accuracy: 0.9634
Epoch 2/10
144/144 [=====] - 27s 188ms/step - loss: 0.0535 - accuracy: 0.9865 - val_loss: 0.1120 - val_accuracy: 0.9729
Epoch 3/10
144/144 [=====] - 28s 194ms/step - loss: 0.0265 - accuracy: 0.9939 - val_loss: 0.0681 - val_accuracy: 0.9799
Epoch 4/10
144/144 [=====] - 27s 188ms/step - loss: 0.0109 - accuracy: 0.9961 - val_loss: 0.0578 - val_accuracy: 0.9843
Epoch 5/10
144/144 [=====] - 28s 192ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.0629 - val_accuracy: 0.9887
Epoch 6/10
144/144 [=====] - 33s 229ms/step - loss: 0.0012 - accuracy: 0.9998 - val_loss: 0.0655 - val_accuracy: 0.9878
Epoch 7/10
144/144 [=====] - 32s 220ms/step - loss: 6.7802e-04 - accuracy: 1.0000 - val_loss: 0.0696 - val_accuracy: 0.9869
Epoch 8/10
144/144 [=====] - 30s 207ms/step - loss: 5.8150e-04 - accuracy: 1.0000 - val_loss: 0.0741 - val_accuracy: 0.9860
Epoch 9/10
144/144 [=====] - 31s 212ms/step - loss: 4.5708e-04 - accuracy: 1.0000 - val_loss: 0.0692 - val_accuracy: 0.9887
Epoch 10/10
144/144 [=====] - 27s 191ms/step - loss: 3.3129e-04 - accuracy: 1.0000 - val_loss: 0.0723 - val_accuracy: 0.9869
<keras.callbacks.History at 0x7fbd75d66a70>
```

shows the binary classification model with early stopping and learning rate reduction on plateau callbacks, achieving accuracy and loss metrics over multiple epochs.

```

loss, accuracy = model.evaluate(X_test, y_test)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)

36/36 [=====] - 1s 34ms/step - loss: 0.1014 - accuracy: 0.9825
Test Loss: 0.10135579109191895
Test Accuracy: 0.9825479984283447

```

3,Transformer based model with BERT-based word embedding.

Define the model architecture:

The model architecture consists of an input layer that takes input_ids and attention_masks as inputs. BERT-based word embeddings are generated using the input layer. Global average pooling is applied to the BERT output, followed by a dense layer with softmax activation for classification. The model summary provides an overview of the model's layers and parameters.


```

Model: "model"
Layer (type)                Output Shape                Param #                    Connected to
-----
input_1 (InputLayer)         [(None, 100)]               0                          []
input_2 (InputLayer)         [(None, 100)]               0                          []
tf_bert_model (TFBertModel)   TFBaseModelOutputWithPooli 109482240                  ['input_1[0][0]',
                        ntions(last_hidden_state=(None, 100, 768),
                        pooler_output=(None, 768),
                        past_key_values=None, hidden_states=None,
                        attentions=None, cross_attentions=None)
global_average_pooling1d (Glob  (None, 768)                0                          ['tf_bert_model[0][0]']
alAveragePooling1D)
dense (Dense)                (None, 3)                  2307                       ['global_average_pooling1d[0][0]']
=====
Total params: 109,484,547
Trainable params: 2,307
Non-trainable params: 109,482,240

```

```

# Load the BERT model
bert_model = TFBertModel.from_pretrained('bert-base-uncased')
# Freeze the BERT layers
bert_model.trainable = False

Downloading tf_model.h5: 100%  536M/536M [00:11<00:00, 48.9MB/s]

```

Load 100%

The model architecture consisted of three epochs of training on the "tweet_airline_sentiment" dataset. The accuracy improved from 25.64% to 46.70% on the training set. However, further enhancements are needed to achieve better performance in sentiment classification.

```

Epoch 1/3
111/111 [=====] - 1733s 15s/step - loss: 1.1392 - accuracy: 0.2564 - val_loss: 1.1109 - val_accuracy: 0.2878
Epoch 2/3
111/111 [=====] - 1678s 15s/step - loss: 1.0800 - accuracy: 0.3521 - val_loss: 1.0559 - val_accuracy: 0.3928
Epoch 3/3
111/111 [=====] - 1669s 15s/step - loss: 1.0304 - accuracy: 0.4670 - val_loss: 1.0097 - val_accuracy: 0.5124
<keras.callbacks.History at 0x7f522519d840>

```

CONCLUSION :

Based on the provided tweet airline sentiment classification results, three models were evaluated: Random Forest Classifier, Multinomial Naive Bayes Classifier, and Support Vector Classifier. Random Forest Classifier achieved an accuracy of 73.57%, precision of 72.61%, recall of 73.57%, and F1-score of 73.57%. In summary, the Random Forest Classifier 74% exhibited the highest performance across all metrics, with the Multinomial Naive Bayes Classifier performing the worst. The Support Vector Classifier performed moderately well, falling between the other two models in terms of accuracy, precision, recall, and F1-score.

The trained model achieved excellent performance on the test set with a loss of 0.0723 and an accuracy of 0.9869. The training process consisted of 10 epochs, during which the model steadily improved. The initial validation loss of 0.1466 reduced to 0.0723 by the final epoch. The accuracy also increased from 0.9634 to 0.9887, indicating the model's ability to correctly classify instances. These results demonstrate the model's capability to effectively learn from the provided data and generalize well to unseen examples. The low loss and high accuracy suggest that the model is reliable and can be utilized for accurate predictions in real-world scenarios.

The transformer-based model with BERT-based word embeddings was trained on the "tweet_airline_sentiment" dataset for three epochs. The model showed improvements in accuracy from 25.64% to 46.70% on the training set. However, there is still room for improvement to achieve better performance on sentiment classification.

GithubLink:- https://github.com/Ranju96/Sentiment_analysis_BiLstm/tree/main