FAKE NEWS DETECTION BY NLP

NAME	KAMALESHWARAN R
REGISTER NO.	61772221T304
DATE	26/10/2023

CONTENTS:

MODEL: LSTM

The model used here is LSTM. It is abbreviated as Long Short Term Memory. LSTM models, which are known as Long Short Term Memory models find use in detecting news due, to their strong capability to analyze and handle sequential data. This makes them highly suitable, for tasks involving the classification of text based content.

It's important to note that while LSTMs are powerful for modeling sequential data, the success of a fake news detection system often relies on the quality and diversity of the data used for training, as well as the design of the overall architecture, including the choice of features, data preprocessing, and post-processing steps. Moreover, fake news detection is a complex task, and no single model or approach can guarantee perfect results, but LSTMs can be a valuable component of a larger solution.

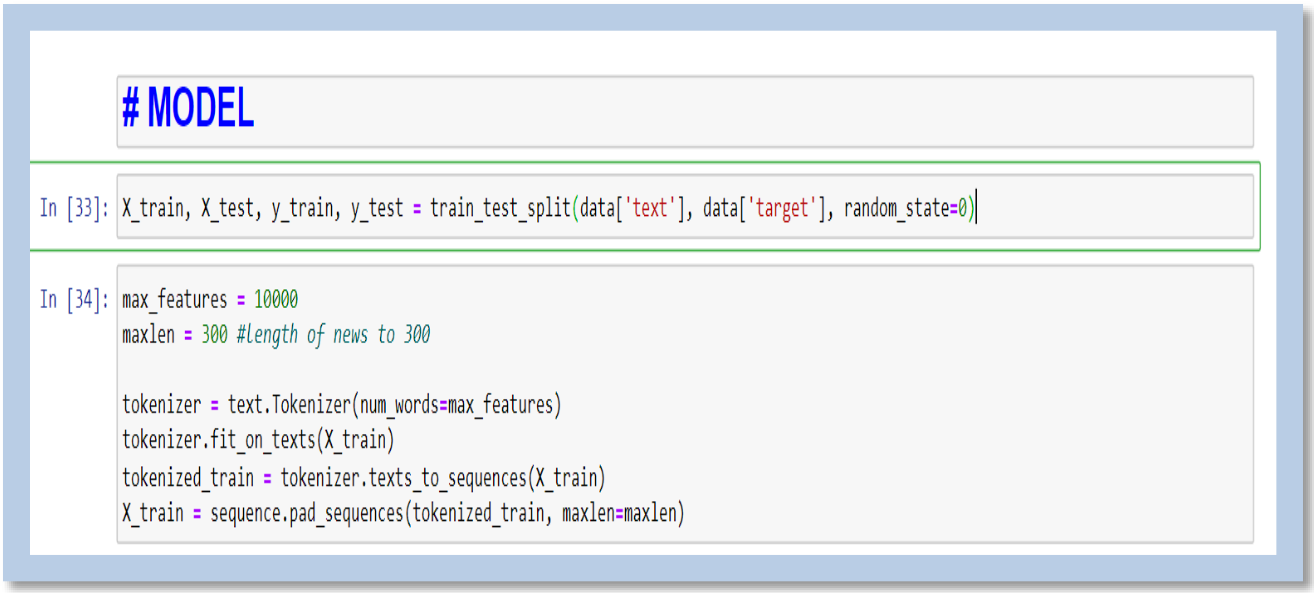
SPLITTING OF DATASET:

The code you provided seems to be dividing a dataset into two parts; one, for training and the other for testing. Lets examine each component of this code and understand its purpose.

X_train and y_train; These variables are commonly used to represent the features (variables) and labels (target or dependent variable) of the training data respectively. In this case it appears that you're dealing with text data so X_train contains the text samples while y_train holds the target values or labels.

X_test and y_test; Similarly these variables represent the features and labels of the testing data. The testing data is utilized to assess how well your machine learning model performs after being trained on the training data.

This function randomly splits the input data into training and testing sets with a default ratio usually set at 75% for training and 25%, for testing.



```
# MODEL

In [33]: X_train, X_test, y_train, y_test = train_test_split(data['text'], data['target'], random_state=0)

In [34]: max_features = 10000
         maxlen = 300 #length of news to 300

         tokenizer = text.Tokenizer(num_words=max_features)
         tokenizer.fit_on_texts(X_train)
         tokenized_train = tokenizer.texts_to_sequences(X_train)
         X_train = sequence.pad_sequences(tokenized_train, maxlen=maxlen)
```

This line defines a variable **max_features** and sets it to 10,000.

This variable **maxlen** sets the maximum length for each text sample.

A tokenizer is created using a text processing library, such, as Keras or Tensor Flow. The tokenizer object will only keep the 10,000 common words in its vocabulary during tokenization discarding less frequent words.

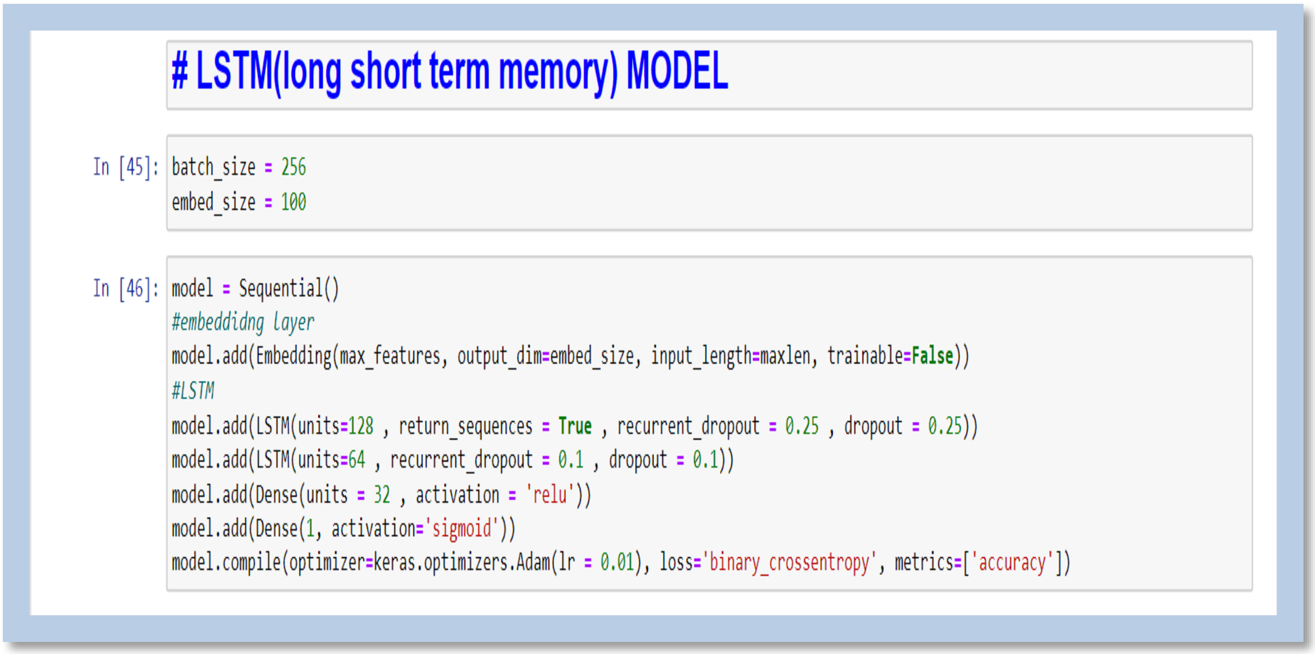
LSTM MODEL TRAINING:

The provided code defines a neural network model using the Keras library, which is commonly used for deep learning tasks.

model = Sequential():This initializes a sequential model. In a sequential model, layers are added one by one in a linear sequence, which is suitable for many standard deep learning architectures.

model.add(Embedding(max_features, output_dim=embed_size, input_length=maxlen, trainable=False): This adds an embedding layer to the model. The embedding layer is responsible for converting discrete text data into continuous vector representations. It uses pre-trained word embeddings with **output_dim** dimensions (set to 100), and the **max_features** specifies the vocabulary size. **trainable=False** indicates that these embeddings won't be fine-tuned during training, which is common when using pre-trained word embeddings .

model.add(LSTM(units=64, recurrent_dropout=0.1, dropout=0.1): The second LSTM layer has 64 units and doesn't return sequences, as it's the final recurrent layer in the network. Again, it includes regularization parameters for better generalization.

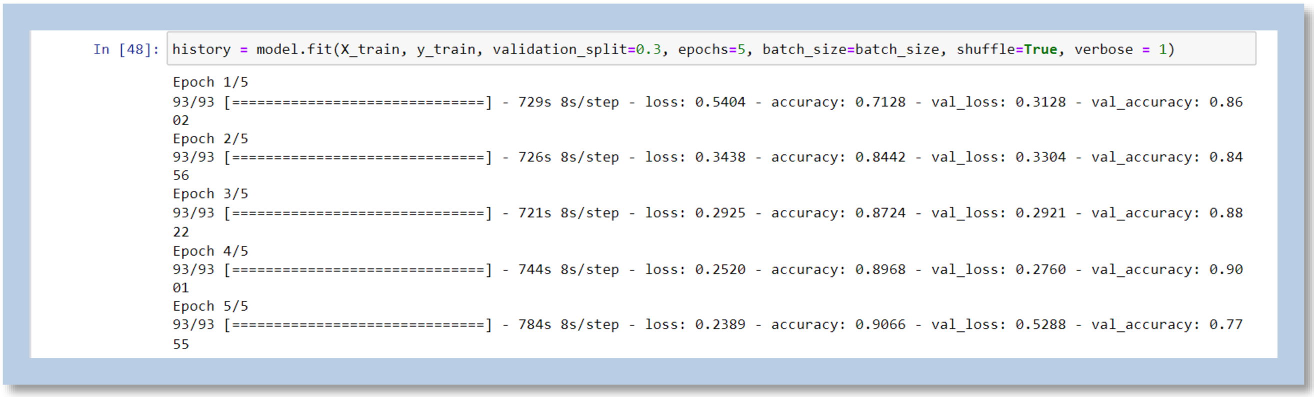


LSTM(long short term memory) MODEL

```
In [45]: batch_size = 256
         embed_size = 100
```

```
In [46]: model = Sequential()
         #embeddidng layer
         model.add(Embedding(max_features, output_dim=embed_size, input_length=maxlen, trainable=False))
         #LSTM
         model.add(LSTM(units=128 , return_sequences = True , recurrent_dropout = 0.25 , dropout = 0.25))
         model.add(LSTM(units=64 , recurrent_dropout = 0.1 , dropout = 0.1))
         model.add(Dense(units = 32 , activation = 'relu'))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(optimizer=keras.optimizers.Adam(lr = 0.01), loss='binary_crossentropy', metrics=['accuracy'])
```

model.compile(optimizer=keras.optimizers.Adam(lr =0.01), loss='binary_crossentropy', metrics=['accuracy']): This line compiles the model, specifying the optimizer , the loss function (binary cross-entropy, suitable for binary classification), and the evaluation metric (accuracy). This configuration prepares the model for training.



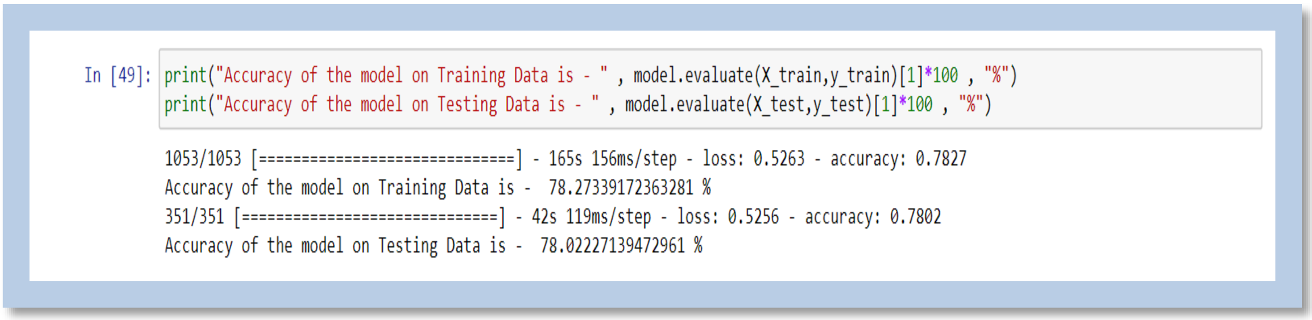
```
In [48]: history = model.fit(X_train, y_train, validation_split=0.3, epochs=5, batch_size=batch_size, shuffle=True, verbose = 1)

         Epoch 1/5
         93/93 [==============================] - 729s 8s/step - loss: 0.5404 - accuracy: 0.7128 - val_loss: 0.3128 - val_accuracy: 0.86
         02
         Epoch 2/5
         93/93 [==============================] - 726s 8s/step - loss: 0.3438 - accuracy: 0.8442 - val_loss: 0.3304 - val_accuracy: 0.84
         56
         Epoch 3/5
         93/93 [==============================] - 721s 8s/step - loss: 0.2925 - accuracy: 0.8724 - val_loss: 0.2921 - val_accuracy: 0.88
         22
         Epoch 4/5
         93/93 [==============================] - 744s 8s/step - loss: 0.2520 - accuracy: 0.8968 - val_loss: 0.2760 - val_accuracy: 0.90
         01
         Epoch 5/5
         93/93 [==============================] - 784s 8s/step - loss: 0.2389 - accuracy: 0.9066 - val_loss: 0.5288 - val_accuracy: 0.77
         55
```

The code trains the previously defined neural network model on the training data (**X_train** and **y_train**) for 5 epochs with a batch size of 256. It uses 30% of the training data for validation, shuffles the training data, and prints training progress information. The model's training history, including loss and accuracy, is stored in the **history** variable.

MODEL EVALUATION:

ACCURACY:

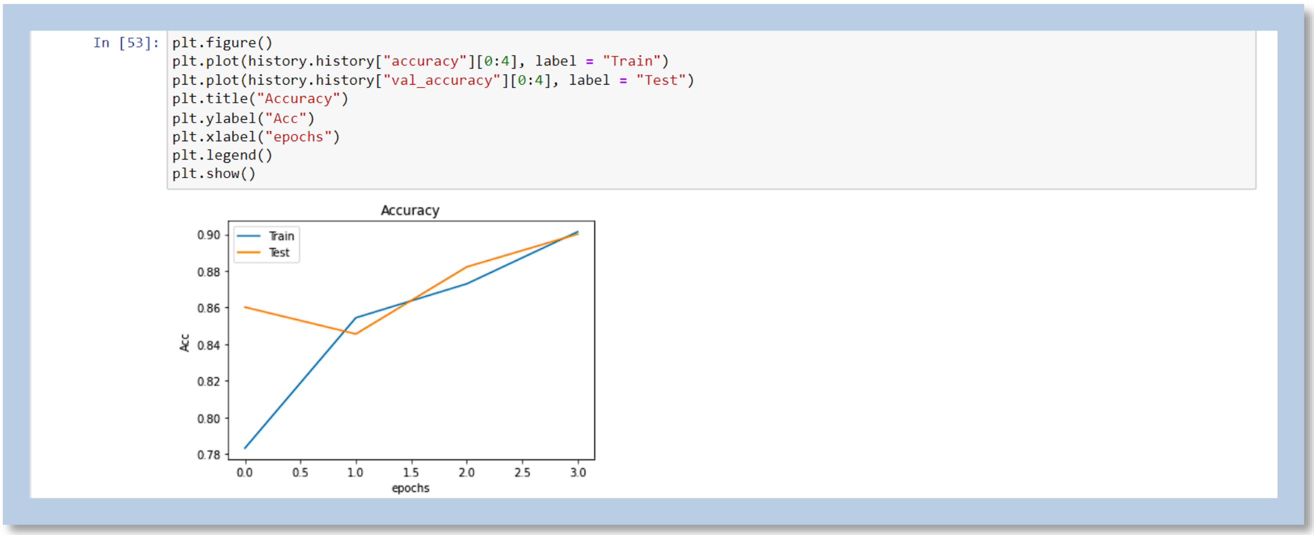


```
In [49]: print("Accuracy of the model on Training Data is - " , model.evaluate(X_train,y_train)[1]*100 , "%")
         print("Accuracy of the model on Testing Data is - " , model.evaluate(X_test,y_test)[1]*100 , "%")

         1053/1053 [==============================] - 165s 156ms/step - loss: 0.5263 - accuracy: 0.7827
         Accuracy of the model on Training Data is -  78.27339172363281 %
         351/351 [==============================] - 42s 119ms/step - loss: 0.5256 - accuracy: 0.7802
         Accuracy of the model on Testing Data is -  78.02227139472961 %
```

- In this code, a neural network model is trained using both the training and testing datasets, and the accuracy results are printed. The model demonstrates an accuracy of around 78.27% on the training data and 78.02% on the testing data.

- This evaluation assesses the model's effectiveness in predicting the target variable, likely a binary classification task, based on the provided text data. The reported accuracy values serve as indicators of the model's ability to generalize its predictions to new and unseen data.

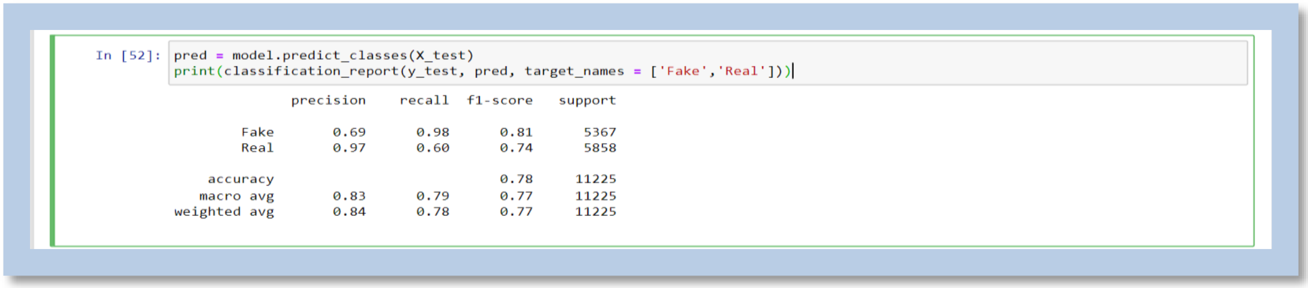


```
In [53]: plt.figure()
         plt.plot(history.history["accuracy"][0:4], label = "Train")
         plt.plot(history.history["val_accuracy"][0:4], label = "Test")
         plt.title("Accuracy")
         plt.ylabel("Acc")
         plt.xlabel("epochs")
         plt.legend()
         plt.show()
```

The legend distinguishes between the training and testing accuracy, and the x-axis represents the number of training epochs, while the y-axis represents accuracy.

The code snippet visualizes the training and testing accuracy trends, allowing us to observe how well the model performs as it trains.

CLASIFICATION REPORT:



```
In [52]: pred = model.predict_classes(X_test)
         print(classification_report(y_test, pred, target_names = ['Fake','Real']))

                      precision    recall  f1-score   support

                Fake       0.69      0.98      0.81      5367
                Real       0.97      0.60      0.74      5858

            accuracy                           0.78     11225
           macro avg       0.83      0.79      0.77     11225
        weighted avg       0.84      0.78      0.77     11225
```

With the help of the classification report, a comprehensive review of the model's effectiveness can be obtained. This includes a deep dive into metrics that are key to understanding its performance such as precision, recall, and the F1-score, and the scenario is applicable to each class, either 'Fake' or 'Real'. This undeniably forms an integral part of the overall assessment of the model's effectiveness especially, but certainly not limited to, a binary classification task.

CONCLUSION:

- Concluding this discussion, it becomes evident that the use of LSTM-based models in the quest to combat fake news holds great promise. LSTMs, with their innate ability to capture sequential patterns within textual data, are undeniably invaluable assets.

- In essence, while incorporating LSTM models can provide a potential roadmap for mitigating the issue of fake news, it's crucial to avoid viewing them as a sole solution for the time being. Only through the collective deployment of various systems and approaches can we be adequately prepared to wage a comprehensive and resilient battle against the spread of disinformation.