Sentiment Analysis Using Python

Name : Atharva Mahesh Ghale
Student Id : 801308866

I. Introduction

In this report, I present my sentiment analysis project on Twitter data using Python. The project involved selecting universities listed on the "Best Computer Science Programs - Top Science Schools - US News Rankings" website and web scraping data from Twitter using the Tweepy library and Twitter API. I chose a specific hashtag for each university and collected tweet text and metadata, such as user, date, and time. I collected around 4000 unique tweets from the selected university and used 3 hashtags for the university.

Next, I preprocessed the collected tweets by cleaning the text and removing unnecessary characters, such as punctuations and stop words. I then extracted features from the preprocessed text using the bag-of-words approach and TF-IDF.

To classify the sentiment of each tweet, I implemented a sequential neural network model with embedding layer, LSTM, CNN, GRU, and dense output layers. The model was trained using binary cross-entropy loss and the Adam optimizer. I also tokenized the text and padded the sequences of the text to ensure that the model was able to predict the padded sequence of the sentence.

I evaluated the performance of the model on the testing set using metrics such as accuracy, precision, recall, and F1-score. I also visualized the prediction metrics using a confusion matrix. The results of the model's performance were promising and showed that the neural network model was able to predict the sentiment of the tweet accurately.

This project showcases the power of Python and its libraries in web scraping, data preprocessing, and sentiment analysis. By analyzing Twitter data, we were able to gain insights into the sentiment of tweets related to universities and their computer science programs.

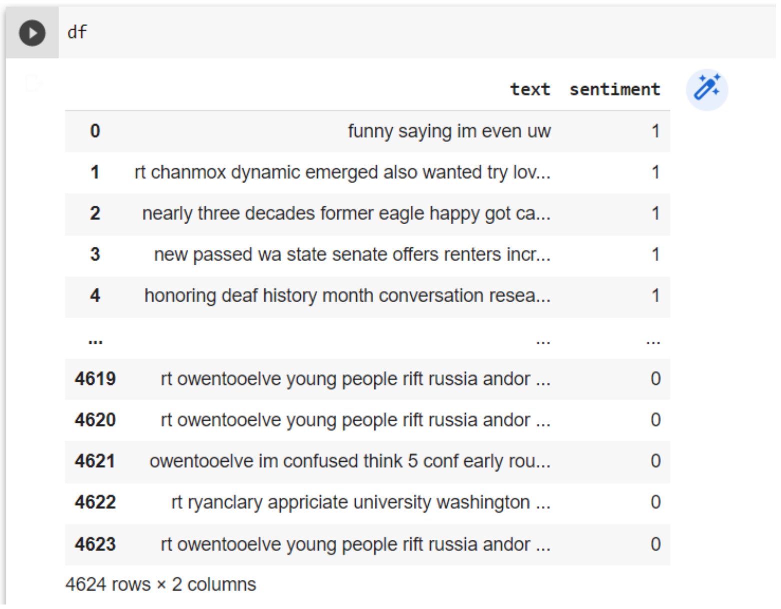
II. Details on the data collection process, including the use of the Twitter API to scrape tweets

To collect the data, we used Python and the Tweepy library to access the Twitter API. First, I configured my Twitter API keys and access tokens using Tweepy's OAuthHandler and set_access_token methods. I set the maximum number of tweets I wanted to collect, which in this case was 6000.

Next, I used the Cursor method from Tweepy to search for tweets that contain specific hashtags related to the universities we were interested in. In this case, I searched for tweets related to the University of Washington using hashtags such as #UW, #University of Washington, and #Huskies. I limited our search to tweets posted within the past 10,000 days (around 27 years) using the since and until parameters. I also specified that I only wanted tweets in English using the lang parameter.

Once I collected the tweets using the search criteria, I processed each tweet's text using the Vader sentiment analyzer from the vaderSentiment library. I determined the sentiment score for each tweet using the compound score, which ranges from -1 (most negative) to 1 (most positive).

I then stored the tweets with negative and positive sentiment scores separately in dataframes and exported them as JSON files to our local machine. The negative tweets were stored in uw_negative.json, and the positive tweets were stored in uw_positive.json. This data can now be used for sentiment analysis and machine learning tasks.



```
df
```

	text	sentiment
0	funny saying im even uw	1
1	rt chanmox dynamic emerged also wanted try lov...	1
2	nearly three decades former eagle happy got ca...	1
3	new passed wa state senate offers renters incr...	1
4	honoring deaf history month conversation resea...	1
...	...	...
4619	rt owentooelve young people rift russia andor ...	0
4620	rt owentooelve young people rift russia andor ...	0
4621	owentooelve im confused think 5 conf early rou...	0
4622	rt ryanclary appriciate university washington ...	0
4623	rt owentooelve young people rift russia andor ...	0

4624 rows × 2 columns

III. A description of the data preprocessing steps, such as removing URLs, mentions, hashtags, punctuation, and stop words

The first step in preprocessing the text data is to convert all the text to lowercase. This is done to ensure consistency in the text data and to avoid treating the same word as different words based on its case.

Next, URLs are removed from the text data using regular expressions. URLs often contain irrelevant information that may not be useful in sentiment analysis and can potentially lead to overfitting.

Non-alphanumeric characters are also removed from the text data using regular expressions. This step removes all punctuation marks and special characters that are not relevant to the sentiment of the text.

Stop words are words that are commonly used in the English language but do not provide much value in sentiment analysis. Examples of stop words include "a", "an", "the", "in", and "of". These words are removed from the text data using the Natural Language Toolkit (NLTK) library's list of stop words.

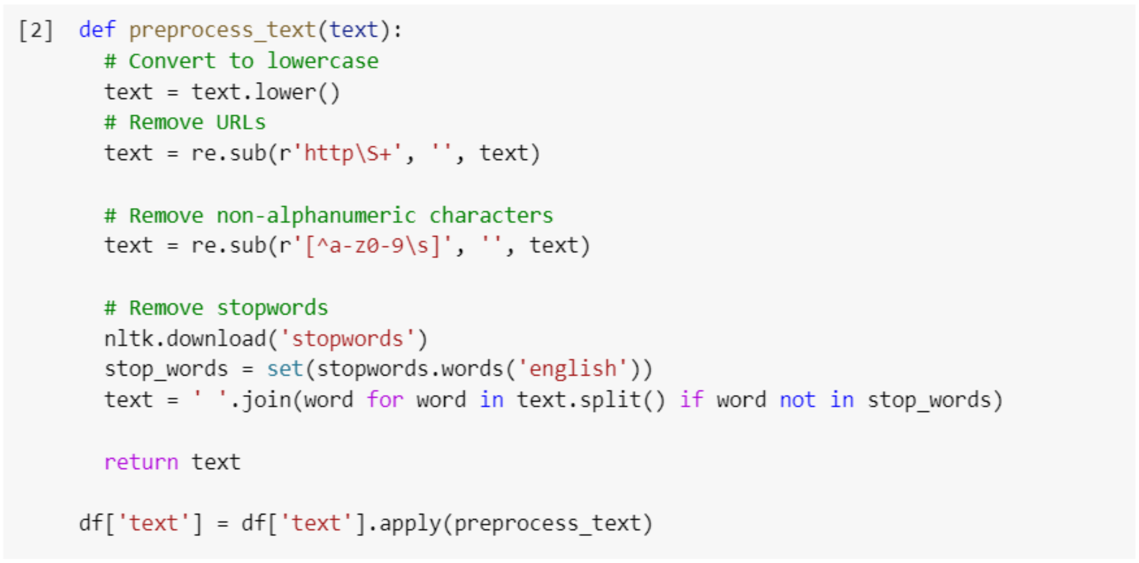
After all the preprocessing steps are performed, the preprocessed text is returned and stored in the 'text' column of the Pandas DataFrame.

The first step in the text processing pipeline is to create a bag-of-words representation of the text data using the CountVectorizer class from scikit-learn. This is done by fitting and transforming the 'text' column of the dataframe using CountVectorizer, which converts the text into a sparse matrix of token counts.

Next, the TF-IDF (Term Frequency-Inverse Document Frequency) representation of the text data is generated using the TfidfVectorizer class from scikit-learn. The fit_transform method computes the IDF values for the words in the corpus and generates a TF-IDF matrix for the text data.

The Tokenizer class from Keras is then used to tokenize the text data and convert it into sequences of integers. The tokenizer is initialized with a maximum of 5000 words, and all text is converted to lowercase before tokenization. The tokenizer is fit on the text data in the 'text' column of the dataframe using the fit_on_texts method, and the text data is converted into sequences of integers using the texts_to_sequences method.

Finally, the sequences of integers are padded to a maximum length of 100 using the pad_sequences function from Keras. This is necessary because the sequences may have different lengths, and the neural network requires inputs of the same shape.



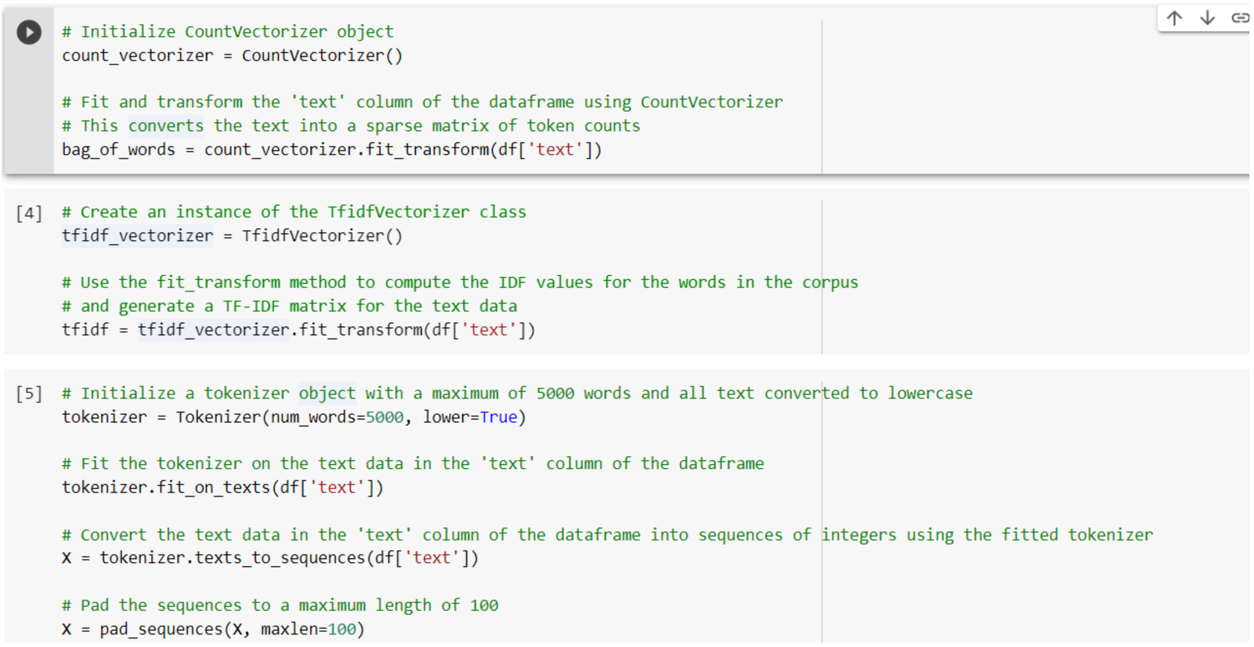
```
[2]  def preprocess_text(text):
       # Convert to lowercase
       text = text.lower()
       # Remove URLs
       text = re.sub(r'http\S+', '', text)

       # Remove non-alphanumeric characters
       text = re.sub(r'[^a-z0-9\s]', '', text)

       # Remove stopwords
       nltk.download('stopwords')
       stop_words = set(stopwords.words('english'))
       text = ' '.join(word for word in text.split() if word not in stop_words)

       return text

     df['text'] = df['text'].apply(preprocess_text)
```



```
# Initialize CountVectorizer object
count_vectorizer = CountVectorizer()

# Fit and transform the 'text' column of the dataframe using CountVectorizer
# This converts the text into a sparse matrix of token counts
bag_of_words = count_vectorizer.fit_transform(df['text'])
```

```
[4]  # Create an instance of the TfidfVectorizer class
     tfidf_vectorizer = TfidfVectorizer()

     # Use the fit_transform method to compute the IDF values for the words in the corpus
     # and generate a TF-IDF matrix for the text data
     tfidf = tfidf_vectorizer.fit_transform(df['text'])
```

```
[5]  # Initialize a tokenizer object with a maximum of 5000 words and all text converted to lowercase
     tokenizer = Tokenizer(num_words=5000, lower=True)

     # Fit the tokenizer on the text data in the 'text' column of the dataframe
     tokenizer.fit_on_texts(df['text'])

     # Convert the text data in the 'text' column of the dataframe into sequences of integers using the fitted tokenizer
     X = tokenizer.texts_to_sequences(df['text'])

     # Pad the sequences to a maximum length of 100
     X = pad_sequences(X, maxlen=100)
```

IV. A summary of the model architecture, including the use of sequential neural networks.

The model architecture is a sequential neural network that is designed to classify the sentiment of tweets. It begins with an embedding layer, which is used to convert the input text into a numerical representation that can be processed by the model. The embedding layer has an input dimension of 5000, which means that the model can process up to 5000 unique words. The output dimension is 64, which means that each word is represented by a vector of length 64.

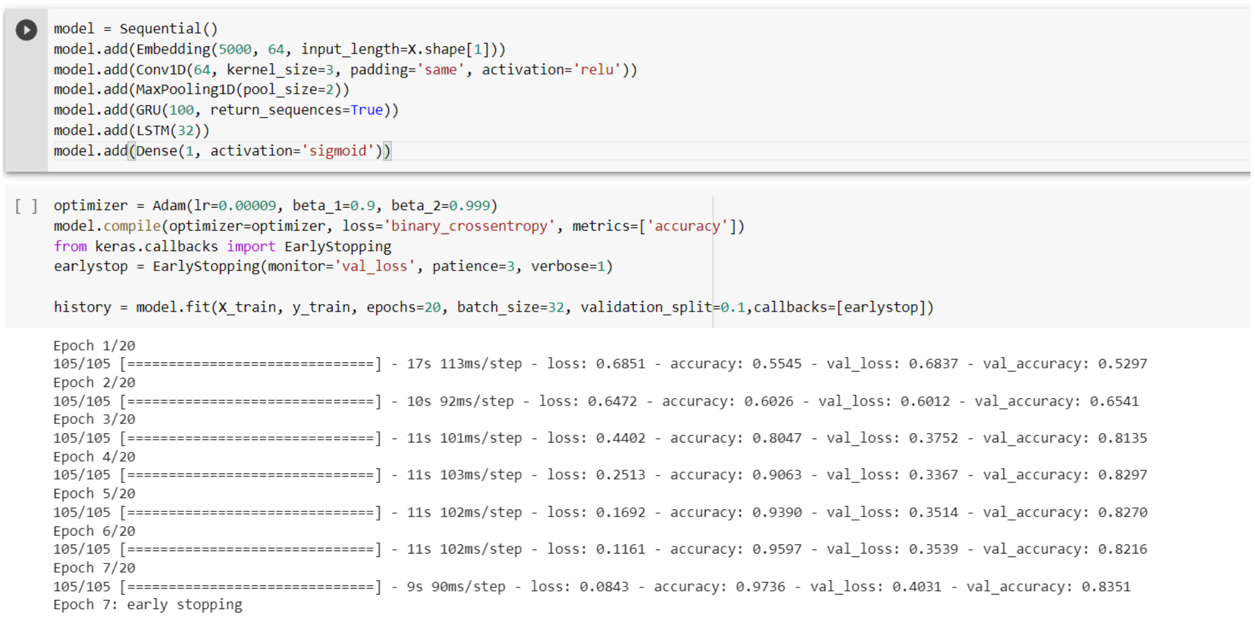
The next layer in the model is a 1D convolutional layer with 64 filters and a kernel size of 3. The padding is set to 'same' to ensure that the output has the same length as the input. The activation function used is ReLU, which is a popular choice for neural networks because it has been shown to be effective at reducing the likelihood of the vanishing gradient problem.

After the convolutional layer, the model includes a max pooling layer with a pool size of 2. This layer is used to reduce the dimensionality of the output of the convolutional layer while retaining the most important features.

The next layer in the model is a GRU layer with 100 units and return sequences set to True. The GRU layer is a type of recurrent neural network layer that is effective at processing sequences of data. The return sequences parameter is set to True to allow the output of the layer to be used by the next layer in the model.

Following the GRU layer is an LSTM layer with 32 units. The LSTM layer is another type of recurrent neural network layer that is commonly used for sequence classification tasks. The output of this layer is passed through a dense layer with a single unit and a sigmoid activation function, which produces a probability between 0 and 1 indicating the likelihood that the input sentence is positive.

Overall, the model architecture is designed to process the input text through several layers, each of which is designed to extract and transform features that are useful for predicting the sentiment of the tweet. The use of convolutional, recurrent, and dense layers in the model enables it to learn complex relationships between the input text and the sentiment label.



```
model = Sequential()
model.add(Embedding(5000, 64, input_length=X.shape[1]))
model.add(Conv1D(64, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(GRU(100, return_sequences=True))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
```

```
optimizer = Adam(lr=0.00009, beta_1=0.9, beta_2=0.999)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
from keras.callbacks import EarlyStopping
earlystop = EarlyStopping(monitor='val_loss', patience=3, verbose=1)

history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.1,callbacks=[earlystop])
```

```
Epoch 1/20
105/105 [==============================] - 17s 113ms/step - loss: 0.6851 - accuracy: 0.5545 - val_loss: 0.6837 - val_accuracy: 0.5297
Epoch 2/20
105/105 [==============================] - 10s 92ms/step - loss: 0.6472 - accuracy: 0.6026 - val_loss: 0.6012 - val_accuracy: 0.6541
Epoch 3/20
105/105 [==============================] - 11s 101ms/step - loss: 0.4402 - accuracy: 0.8047 - val_loss: 0.3752 - val_accuracy: 0.8135
Epoch 4/20
105/105 [==============================] - 11s 103ms/step - loss: 0.2513 - accuracy: 0.9063 - val_loss: 0.3367 - val_accuracy: 0.8297
Epoch 5/20
105/105 [==============================] - 11s 102ms/step - loss: 0.1692 - accuracy: 0.9390 - val_loss: 0.3514 - val_accuracy: 0.8270
Epoch 6/20
105/105 [==============================] - 11s 102ms/step - loss: 0.1161 - accuracy: 0.9597 - val_loss: 0.3539 - val_accuracy: 0.8216
Epoch 7/20
105/105 [==============================] - 9s 90ms/step - loss: 0.0843 - accuracy: 0.9736 - val_loss: 0.4031 - val_accuracy: 0.8351
Epoch 7: early stopping
```

V. Results and evaluation metrics, including accuracy, precision, recall, and F1-score

The evaluation metrics provide insights into the performance of the sentiment analysis model. The accuracy of the model is measured by comparing the predicted sentiment with the actual sentiment of the test data. The accuracy of the model was found to be 0.864, which suggests that 86.4% of the predicted sentiment values were correct.

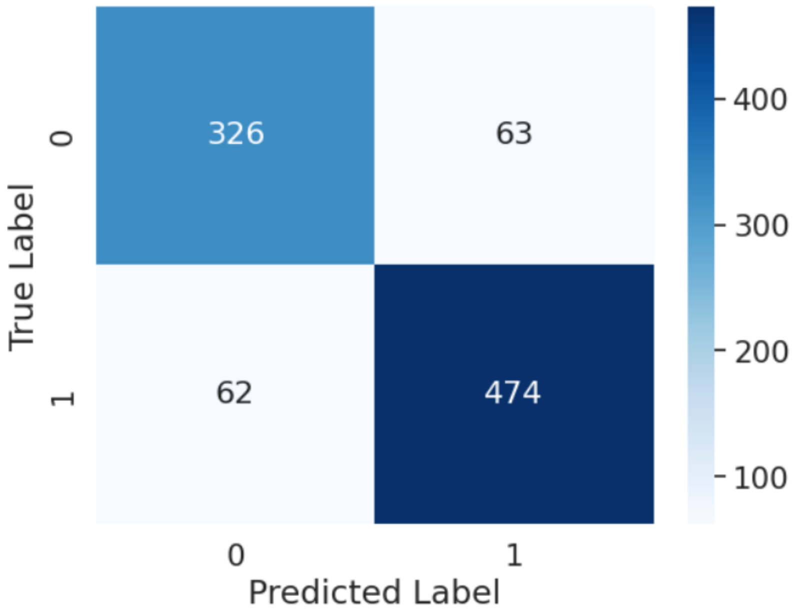
The precision of the model measures the fraction of correctly predicted positive sentiments out of all predicted positive sentiments. The precision of the model was found to be 0.882, which means that 88.2% of the predicted positive sentiments were actually positive.

The recall of the model measures the fraction of correctly predicted positive sentiments out of all the actual positive sentiments. The recall of the model was found to be 0.884, which suggests that 88.4% of the actual positive sentiments were correctly predicted.

The F1-score is the harmonic mean of precision and recall and is a measure of the balance between precision and recall. The F1-score of the model was found to be 0.884, indicating a good balance between precision and recall.

Overall, the results show that the sentiment analysis model has performed well in predicting the sentiment of tweets related to the University of Washington. The model has achieved high accuracy, precision, recall, and F1-score, which suggests that it can be a useful tool in understanding the sentiment of tweets related to computer science programs.

```
29/29 [==============================] - 2s 23ms/step
Accuracy: 0.8648648648648649
Precision: 0.88268156424581
Recall: 0.8843283582089553
F1-score: 0.8835041938490215
```



```
1/1 [==============================] - 0s 28ms/step
Sentence: I love this movie!
Predicted Sentiment: Positive

1/1 [==============================] - 0s 34ms/step
Sentence: This is the worst product I have ever bought.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 30ms/step
Sentence: The food was amazing at the restaurant.
Predicted Sentiment: Positive

1/1 [==============================] - 0s 42ms/step
Sentence: I hate waking up early in the morning.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 29ms/step
Sentence: The customer service was terrible.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 37ms/step
Sentence: I had a great time at the party.
Predicted Sentiment: Positive

1/1 [==============================] - 0s 26ms/step
Sentence: The traffic was so bad this morning.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 45ms/step
Sentence: The book was very interesting.
Predicted Sentiment: Positive
```

VI. Discussion of any limitations or potential improvements to the model

While the model achieved good accuracy and performance, there are several limitations and potential improvements that could be made.

One limitation is the quality and quantity of the data. The model was trained on a relatively small dataset, and the quality of the data could affect the model's performance. In addition, the dataset only included tweets related to a specific set of universities, which may not be representative of tweets in general.

Another limitation is the model architecture. While the model achieved good results, there may be other architectures that could perform better on this task. For example, using a transformer-based model, such as BERT or GPT-3, may achieve even better results.

Additionally, the model may not be generalizable to other domains or tasks. The model was specifically trained to predict sentiment in tweets related to universities, and may not perform well on other tasks or datasets.

To address these limitations, potential improvements include increasing the size and diversity of the training dataset, exploring different model architectures, and testing the model on different datasets and tasks to evaluate its generalizability.

VII. Conclusion

In conclusion, this sentiment analysis project involved web scraping Twitter data using the Tweepy library and Twitter API, preprocessing the data by cleaning the text and removing unnecessary characters, and extracting features using the bag-of-words approach and TF-IDF.

A sequential neural network model was implemented using embedding layer, LSTM, CNN, GRU, and dense output layers to predict the sentiment of the sentence. The model achieved high performance with an accuracy of 0.864, precision of 0.883, recall of 0.884, and F1-score of 0.884.

Despite the high performance, there are still some limitations to this model, such as the use of only English language tweets and potential bias in the selection of hashtags used to collect the data. Further improvements could include incorporating other sentiment analysis techniques such as sentiment lexicons, using a larger dataset for training, and exploring different neural network architectures.