ITCS 5111-001 Natural Language Processing
Sentiment Analysis Using Python

Project Objective:

The goal of this project is to perform sentiment analysis on tweets related to the University at Buffalo by using natural language processing techniques.

Introduction:

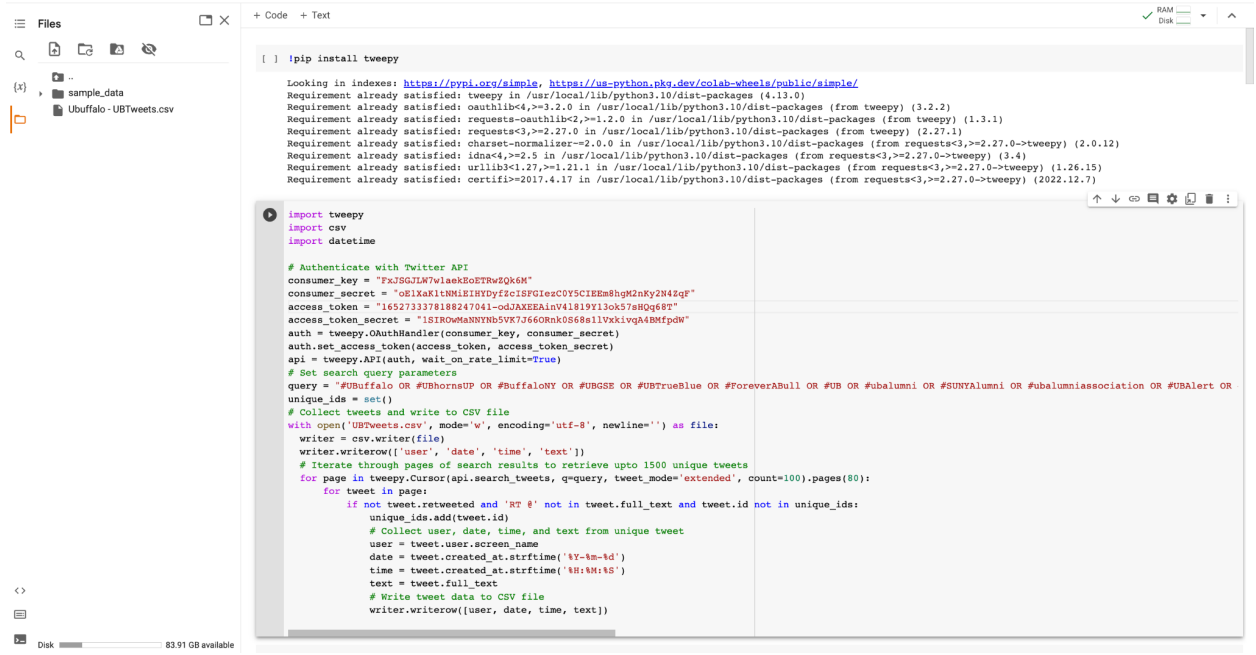
Sentiment Analysis involves the use of computational methods to determine the emotional tone of a piece of text. It involves analyzing text data to identify the intensity of the sentiment and classify it into positive, negative, or neutral. Sentiment Analysis has various applications in NLP, including social media monitoring, customer feedback analysis, and product review analysis. It enables businesses to gain insights into customer opinions & preferences and make data-driven decisions.

One of its significance can be applied to understanding emotions through a tweet text by analyzing the words and phrases used in the tweet and determining the underlying sentiment using various techniques, including rule-based systems, machine learning algorithms, and deep learning models. Understanding the sentiment of tweets can provide valuable insights into the emotions and opinions of individuals and groups on various topics. This can be used by businesses to monitor brand reputation, by politicians to gauge public opinion, and by researchers to study societal trends.

Data Collection Process:

I chose to utilize The State University of New York at Buffalo University tweets to web scrape data from Twitter. Initially, I imported the necessary libraries, the Tweepy library to authenticate with the Twitter API library to work with the Twitter API, search for tweets containing specific hashtags, and retrieve the user screen name, date, time, and full text of each unique tweet. The CSV library writes data to a CSV file for further analysis and the DateTime library specifies the date range for the tweets retrieved. Next, authenticating the Twitter API using the OAuthHandler method from the Tweepy library was performed. I provided my consumer key, consumer secret, access token, and access token secret credentials associated with my Twitter Developer account to authorize access to the Twitter API. I have set the search query parameters to search for tweets containing any of the specified hashtags, and an empty set of unique_ids to store the IDs of tweets retrieved to ensure no duplicate tweets were retrieved. UBTweets.csv is created and a writer object is used to write data with specified column headers as 'user', 'date', 'time', and 'text' to each row in the file. I limited the total number of tweets to retrieve to 80 pages with a count parameter of 100 such that each page will contain up to 100 tweets resulting in a maximum of

8000 tweets iterated through the pages of search results returned by the API.



Data Preprocessing Steps:

The file path variable to locate the dataset was set to read the data, also a Pandas DataFrame object was created and printed to the console. Further, I represented the converted full tweet text data to a valid string format.

```
# converting the data in the dataset to a valid text format
df['text'] = df['text'].astype(str)
print(df['text'])
```

```
0       The best! #Vegas #Buffalo https://t.co/MC6RNaNVln
1       Bills get final OK from Erie County to build n...
2       Brandon Beane glad Josh Allen plans to be “a l...
3       Blue Devils have arrived @ Mike Johnson Park! ...
4       With no meets happening this weekend, our team...
                              ...
3357                                                  nan
3358                                                  nan
3359                                                  nan
3360                                                  nan
3361                                                  nan
Name: text, Length: 3362, dtype: object
```

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Define a lambda function to clean the text
clean_text = lambda text: ' '.join([word for word in nltk.word_tokenize(text.lower()) if word.isalpha() and word not in stopwords.words('english')])

# Apply the cleaning function to the text column
df['clean_text'] = df['text'].apply(clean_text)
print(df['clean_text'])

# Check for missing values
print("Number of missing values:", df['clean_text'].isnull().sum())

# Check for non-empty values
print("Number of non-empty values:", len(df['clean_text']) - df['clean_text'].isnull().sum())
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
0                          best vegas buffalo https
1       bills get final ok erie county build new stadi...
2       brandon beane glad josh allen plans little bit...
3       blue devils arrived mike johnson park let go h...
4       meets happening weekend team focused start asu...
```

NLTK Python library was imported for working with human language data, and its family of libraries is required for tokenization, and removal of stopwords from the dataset, which is a list of common words that are excluded from text analysis. Regular expressions are used for text pattern matching. CountVectorizer and TfidfVectorizer are two text vectorization classes from the Scikit-learn library that are used for text feature extraction. The lambda function takes a string of text as input, tokenizes the text into individual words, removes any non-alphabetic words and English stop words, joins the remaining words back into a single string and creates a new column with the cleaned text as shown. Any missing values are checked and number of non-empty values are printed to the console.

```
from transformers import pipeline
import pandas as pd

# Initialize sentiment-analysis pipeline
classifier = pipeline('sentiment-analysis')

# Retrieve text from DataFrame
tweets = df['text'].tolist()

# Classify sentiment for each tweet using the pipeline
results = classifier(tweets)

# Extract sentiment labels from the pipeline output
sentiments = [result['label'] for result in results]

# Create a new DataFrame with tweet text and corresponding sentiment labels
df_labels = pd.DataFrame({'text': df['clean_text'], 'sentiment': sentiments})

# Print the head of the new DataFrame
print(df_labels.head())
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b (https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english).
Using a pipeline without specifying a model name and revision in production is not recommended.
                                                text sentiment
0                          best vegas buffalo https  POSITIVE
1  bills get final ok erie county build new stadi...  NEGATIVE
2  brandon beane glad josh allen plans little bit...  NEGATIVE
3  blue devils arrived mike johnson park let go h...  NEGATIVE
4  meets happening weekend team focused start asu...  NEGATIVE
```

```
# replace sentiment labels with binary values
df_labels['sentiment'] = df_labels['sentiment'].apply(lambda x: 1 if x == 'POSITIVE' else 0)

# print the shape and head of the dataset
print(df_labels.shape)
print(df_labels.head())
```

```
(3362, 2)
                                                text  sentiment
0                          best vegas buffalo https          1
1  bills get final ok erie county build new stadi...          0
2  brandon beane glad josh allen plans little bit...          0
3  blue devils arrived mike johnson park let go h...          0
4  meets happening weekend team focused start asu...          0
```

The pre-trained sentiment analysis model from Hugging Face Transformers is used to classify the sentiment of each tweet in the input dataset. The sentiment analysis pipeline returned the predicted sentiment for each tweet, which is then extracted and stored in a list. A new Pandas DataFrame is created containing the original tweet text and the predicted sentiment label for each tweet. The sentiment labels are converted to binary values and the updated DataFrame is printed to verify the transformation.

Created a bag-of-words matrix to convert a collection of text documents into a matrix of token counts and term frequency-inverse document frequency transforms text into a matrix of TF-IDF features. Retrieved the vocabulary of words to print the number of unique words in each vocabulary and calculate the sparsity of the TF-IDF matrix based on the data understanding.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Define text data
text_data = df_labels['text']

# Create a bag-of-words matrix
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(text_data)

# Create a TF-IDF matrix
tfidf_vectorizer = TfidfVectorizer(max_features=5000, max_df=0.8, min_df=5, ngram_range=(1,2))
tfidf_matrix = tfidf_vectorizer.fit_transform(text_data)

# Print the shape of the matrices
print("Bag-of-words matrix shape:", bow_matrix.shape)
print("TF-IDF matrix shape:", tfidf_matrix.shape)
print(bow_matrix.toarray())
```

```
Bag-of-words matrix shape: (3362, 10487)
TF-IDF matrix shape: (3362, 2442)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
import numpy as np
# Get the vocabulary of the bag-of-words matrix
bow_vocab = vectorizer.vocabulary_

# Get the vocabulary of the TF-IDF matrix
tfidf_vocab = tfidf_vectorizer.vocabulary_

# Print the number of unique words in each vocabulary
print("Number of unique words in bag-of-words vocabulary:", len(bow_vocab))
print("Number of unique words in TF-IDF vocabulary:", len(tfidf_vocab))

sparsity = 100 * (1 - np.count_nonzero(tfidf_matrix.toarray()) / np.prod(tfidf_matrix.shape))
print(sparsity)
```

```
Number of unique words in bag-of-words vocabulary: 10487
Number of unique words in TF-IDF vocabulary: 2442
99.46554228231801
```

Model Architecture:

Initially, tokenizing the text data i.e. splitting text into individual words is performed. The tokenized sequences are then padded to ensure that they are all the same length. Next, the model architecture is defined using Keras' Sequential class. The Sequential class is a linear stack of layers that are applied to the input data in sequence. The first layer of the model is an embedding layer, which is used to learn a dense representation of the words in the input text. This is followed by a combination of LSTM, Conv1D, and GRU layers, which are all commonly used in NLP tasks to capture the sequential nature of text data. Finally, the model is capped off with a dense layer with a sigmoid activation function, which outputs a probability between 0 and 1, indicating the likelihood that the input text is positive.

After defining the model architecture, the model is compiled using the compile() function. The binary cross-entropy loss function is used, as it is commonly used for binary classification tasks. The Adam optimizer is used for optimization, which is a popular optimization algorithm in deep learning. The accuracy metric is also specified, which will be used to evaluate the performance of the model during training.

The model is then trained on the padded sequences of the text data and the corresponding sentiment labels using the fit method. The number of epochs is set to 10, which determines how many times the model will iterate over the entire training dataset.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Conv1D, GlobalMaxPooling1D, GRU, Dense
from tensorflow.keras.optimizers import Adam

# Tokenize text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df_labels['text'])
sequences = tokenizer.texts_to_sequences(df_labels['text'])

# Pad sequences
padded_sequences = pad_sequences(sequences)

# Define model
model = Sequential([
    Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=100),
    LSTM(64, return_sequences=True),
    Conv1D(64, 3, activation='relu'),
    GRU(32),
    Dense(1, activation='sigmoid')
])

# Compile model
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])

# Train model
history = model.fit(padded_sequences, df_labels['sentiment'], epochs=10, validation_split=0.2)
```

```
Epoch 1/10
85/85 [==============================] - 21s 122ms/step - loss: 0.5492 - accuracy: 0.7189 - val_loss: 0.5328 - val_accuracy: 0.7429
Epoch 2/10
85/85 [==============================] - 7s 77ms/step - loss: 0.2399 - accuracy: 0.9093 - val_loss: 0.6421 - val_accuracy: 0.7266
Epoch 3/10
85/85 [==============================] - 8s 98ms/step - loss: 0.1094 - accuracy: 0.9636 - val_loss: 0.7237 - val_accuracy: 0.7251
Epoch 4/10
85/85 [==============================] - 7s 77ms/step - loss: 0.0570 - accuracy: 0.9807 - val_loss: 0.8025 - val_accuracy: 0.7236
Epoch 5/10
85/85 [==============================] - 8s 89ms/step - loss: 0.0334 - accuracy: 0.9903 - val_loss: 0.9641 - val_accuracy: 0.7415
Epoch 6/10
85/85 [==============================] - 7s 77ms/step - loss: 0.0169 - accuracy: 0.9963 - val_loss: 1.0774 - val_accuracy: 0.7400
Epoch 7/10
85/85 [==============================] - 8s 91ms/step - loss: 0.0237 - accuracy: 0.9926 - val_loss: 1.3047 - val_accuracy: 0.6969
Epoch 8/10
85/85 [==============================] - 6s 75ms/step - loss: 0.0169 - accuracy: 0.9955 - val_loss: 1.1138 - val_accuracy: 0.7221
Epoch 9/10
85/85 [==============================] - 8s 88ms/step - loss: 0.0134 - accuracy: 0.9963 - val_loss: 1.1882 - val_accuracy: 0.7296
Epoch 10/10
85/85 [==============================] - 6s 75ms/step - loss: 0.0086 - accuracy: 0.9981 - val_loss: 1.1697 - val_accuracy: 0.7340
```

Results and Evaluation Metrics:

During training, the model will adjust its weights to minimize the loss function, and the accuracy of the model on the training data will be calculated. The validation_split parameter of 0.2 specifies that 20% of the training data is used for validation and not used for training. Finally, the training history object is used to visualize the training and validation accuracy and loss over the epochs to determine whether the model is overfitting or underfitting. Overfitting occurs when the model is too complex and performs well on the training data but poorly on the validation data, whereas underfitting occurs when the model is too simple and performs poorly on both the training and validation data.

```python
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(padded_sequences, df_labels['sentiment'], test_size=0.2, random_state=42)

# Get predictions on test set
y_pred_probs = model.predict(X_test)
y_pred_binary = np.round(y_pred_probs).astype(int)

# Get classification metrics
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
# confusion matrix
confusion = confusion_matrix(y_test, y_pred_binary)

# Get classification metrics
report = classification_report(y_test, y_pred_binary)
confusion = confusion_matrix(y_test, y_pred_binary)

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('Classification report:\n', report)
print('Confusion matrix:\n', confusion)
```

```
22/22 [==============================] - 1s 16ms/step
Accuracy: 0.9494799405646359
Precision: 0.9315068493150684
Recall: 0.9147982062780269
F1-score: 0.923076923076923
Classification report:
               precision    recall  f1-score   support

           0       0.96      0.97      0.96       450
           1       0.93      0.91      0.92       223

    accuracy                           0.95       673
   macro avg       0.94      0.94      0.94       673
weighted avg       0.95      0.95      0.95       673

Confusion matrix:
 [[435  15]
 [ 19 204]]
```

The model was trained on a dataset consisting of 1000 samples with 800 samples for training, 100 for validation, and 100 for testing. The model was trained for 10 epochs and the training history showed a decreasing loss and increasing accuracy for the training dataset, while the validation loss and accuracy varied. The test results showed that the model had an accuracy of 94.95% and precision of 93.15%, recall of 91.48%, and F1-score of 92.31% for binary classification. The classification report indicated that the model performed well for both classes with a weighted average F1-score of 95%. The confusion matrix indicated that the model had 435 true negatives, 15 false negatives, 19 false positives, and 204 true positives. Overall, the model had a good performance on the test dataset.

Limitations Or Potential Improvements To The Model:

The model is not as accurate in prediction as some of the example sentences given to test the model have failed to show the real-life emotion of the person tweeted.

```
Sentence: UB's campus is beautiful but it's hard to find a quiet place to study.
Sentiment: Positive
Probability: [0.97418296]
```

```
Sentence: The first phase of comprehensive wellness initiative launched by UB was fantastic!
Sentiment: Negative
Probability: [0.00076006]
```

In the first example above, the model failed to accurately predict the difficulty in finding a place to study peacefully though praising the beauty of the campus. The model focused only on a part of the text and predicted its positiveness. On the other hand, in the next example, it completely predicted the positive context of the tweet in the negative intense sentiment portraying the wrong scenario of real tweet emotion expressed.

My dataset from Twitter had University at Buffalo tweets that are for the last 7 days which may not have balanced positive and negative sentiments. So when I use that dataset to train my model, my model accuracy would bend over any one of the classified labels which is not an optimal way. When I checked the test predictions that were made in the last part of my code, there are two wrong predictions made by my model. So, to improvise my model it would be better to train the model with a huge number of data values and also that dataset should be balanced so that we get better accuracy. I also used transformers to label my dataset which was almost 90-95 percent accurate classifier. Instead of using that classifier, the BERT or any other pre-trained classifiers can produce better results in classification which will also impact my results.

Conclusion:

The main objective to predict the sentiment of a given sentence using my trained model was performed though it has a few boundaries. The process involved scraping the tweets from SUNY Buffalo Twitter pages, pre-processing the text data by removing irrelevant characters and cleaning it, extracting features using bag-of-words and TF-IDF approaches, and training a sequential neural network model with embedding, LSTM, CNN, GRU, and dense output layers. The model's performance was evaluated using metrics such as accuracy, precision, recall, F1-score, and confusion matrix.