Natural Language Processing (ITCS 5111) Project

Sentimental Analysis Using Python

Student Name – Boyidapu Nikhita Sai

Student Id - 801327682

Introduction

This project demonstrates the use case of sentimental analysis using Natural Language Processing techniques using the tweets scraped from Twitter by providing various hashtags related to **University of South Florida**. The tweets are scraped from Twitter using Twitter API and Tweepy library. It extracts information such as tweet text, username, created date and time. Once the data is fetched, the raw data is processed and then cleaned by removing unnecessary characters and URLs. The features were extracted using Count vectorizer and TF-IDF approach and then RoBERTa pre-trained model is used for labelling the sentiment for each text.

A sequential neural network model is implemented using various layers such as embedding, LSTM, CNN, GRU, and dense output layers for training the model using binary cross-entropy loss and the Adam optimizer. Lastly, the performance of the model on the testing set using evaluation metrics such as accuracy, precision, recall, and F1-score

The main objective of this project is to create a neural network model for sentimental analysis for predicting the sentiment of the text.

Data Collection

For this project, I chose University of South Florida for fetching the tweets related to that university. These tweets are fetched using Tweepy library and Twitter API. This data contains information such as tweet text, username, created date and time. The prerequisites for this step are having Twitter Developer account and installing Tweepy library. I created one Project and App in Twitter Developer Portal. I extracted Twitter API keys such as API Tokens, API Secret, Access Token and Access Secret. These API keys are required to be given for authentication. Once, the authentication is completed. Hashtags are saved in a list and using **tweepy.cursor()** function, each hashtag was sent as a query and am fetching 2000 tweets for each. Since, the tweets in the dataset should be unique, therefore, before adding the entire metadata of tweet, I added a validation for saving tweets text in a set and checking it for every iteration. This prevents the dataset from having duplicates. The final set named **uniquetweetlist** containing the metadata is saved in a csv file for further process.

```
auth = tweepy.OAuthHandler("PAjwwvpWPw9l308c5I7LTWiAU",
    "i660c0iZSsVnFPM0OowjWAsW4j3HGtcU35Xl0B8Ev9GkA2uxDk")
auth.set_access_token("1649879424311885824-QJSQCgLX3UPETwrGV0IthFS9ug6Vq5",
    "80wPQa5TdgAeOenzxAzDyu0XkpakWNL7V31LRRirTAIlg")

api = tweepy.API(auth)

try:
    api.verify_credentials()
    print("Authentication OK")
except:
    print("Error during authentication")
```

```
Authentication OK
```

```
hashtags = ["#usf","#BullPride","#USF","#usfbulls","#USFGives","#HornsUp","#ComeToTheBay","#ComeToTheBay","#StayInTheBay","#ProBu
uniquetweetlist=set()
tweetlist=set()
for hashtag in hashtags:
    for tweet in tweepy.Cursor(api.search_tweets, q=hashtag, lang="en", tweet_mode="extended").items(2000):
        if tweet.full_text not in tweetlist:
            uniquetweetlist.add(tweet)
            tweetlist.add(tweet.full_text)
len(uniquetweetlist)
```

```
with open(f"USFFinalDataset.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow(["Text", "User", "Date", "Time"])
    for eachtweet in uniquetweetlist:
        writer.writerow([eachtweet.full_text, eachtweet.user.screen_name, eachtweet.created_at.date(), eachtweet.created_at.time(
```

Data Pre-processing

In this step, the raw data is given as input for processing and cleaning the data using nltk and regex techniques. The data cleaning removes unwanted characters and stopwords from the raw data. This helps in removing noise in the dataset.

The data cleaning steps includes:

1. **Converting to lower case:** All the words are converted into lower case using lower() function, which helps in better performance of the model.
2. **Removing URLs, Mentions, Hashtags, Retweet tags:** Using regex functions, the URLs and websites are identified by **https?** and **www** patterns and are replaced by whitespaces. Similarly, the words starting with **@** and **#** are considered as mentions and hashtags respectively. For every retweeted tweet, **RT** is displayed at the beginning of the text, therefore it is replaced with spaces.
3. **Removing non-alpha characters:** All the words, other than those containing alpha characters at the beginning are removed.
4. **Removing stopwords:** The text is tokenized into words and all the stopwords were removed from the text using stopwords, extracted from nltk corpus.

Data Preprocessing

```
import re
from nltk.corpus import stopwords
def preprocess_text(text):
    text = text.lower()
    text = re.compile(r'https?://\S+|www\.\S+|@[A-Za-z0-9]+|#[A-Za-z0-9]+|rt').sub('', text)
    text = re.sub('[^a-z A-Z]+', ' ', text)
    text = text.strip()
    text = ' '.join([word for word in text.split() if word.lower() not in stop_words])
    return ''.join(text)
stop_words = set(stopwords.words('english'))
datafile['Text'] = datafile['Text'].astype(str).apply(preprocess_text)
```

```
datafile['Text'][3]
```

```
'congratulations usf pasco pds network teacher leader academy coho graduates excited see amaz'
```

Feature Extraction

Feature Extraction is used for transforming text into numerical representations for passing it as input for ML models. For this, I used Count Vectorizer, TF-IDF Transformer. **Count Vectorizer** is used for creating word count matrix for each word in the text. In the matrix, row is represented as "text" and column represents "unique word". **TF-IDF Transformer** is used for weighing the importance of each word by taking the input matrix we got from count vectorizer matrix. The weight depends on the frequency of the word. The higher the frequency, the more weight for that particular word.

In the below code, the text is given input for count vectorizer function and is transformed into the matrix. The matrix is converted into the binary array and is given as input for TF-IDF transformer for weighing the importance for each word.

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(datafile['Text'])
word_counts = X.toarray()
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(word_counts)
print("Word counts matrix:", X)
print("TF-IDF matrix:", tfidf)
```

The sentiment for the text in the datafile can be predicted using transformers. There are numerous pre-trained transformers that can be used for sentimental analysis such as BERT, Albert, RoBerta, pipeline etc. These transformers consists of pre-trained model, which will predict the sentiment for each text and return the polarity of the text in the labels as -1, 0, 1 representing negative, neutral, positive. In this code, I used **RoBerta** transformer for finding the sentiment. RoBerta is the extension of BERT transformer. **twitter-roberta-base –sentiment** is a pre-trained model which is already trained and used for sentiment analysis. With the help of this model, I am labelling the text with sentiments values. Each text in the Text column from the dataset is passed as input for **sentiment_prediction** function. The text is tokenized and sequences were padded using the pre-tained tokenizer within the model. The padded sequence is passed as input for the model and the output were returned as three-indexed labels in logits. From the logits, the maximum value index is taken using **argmax()** function.

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
model_name = "cardiffnlp/twitter-roberta-base-sentiment"
model = AutoModelForSequenceClassification.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
def sentiment_prediction(tweet):
    print(tweet)
    inputs = tokenizer(tweet, padding=True, truncation=True, max_length=512, return_tensors="pt")
    outputs = model(**inputs)
    label = outputs.logits.argmax().item() - 1
    return label
```

```
datafile['Sentiment Value'] = datafile['Text'].apply(lambda x: sentiment_prediction(x[:512]))
```

Model Architecture

For training the model, the dataset is divided into 80% trained data and 20% test data. This model is implemented using sequential neural network. The training text data is preprocessed using a Tokenizer and converted to sequences using the texts_to_sequences() method.

The model architecture of this model consists of different types of layers such as embedding layer, LSTM, CNN, GRU, and dense output layers.

Embedding layer: This layer is used for mapping each word to a fixed-sized vector and will create a dense representation for the input sequence. The layer is taking the input dimension as 5000 representing the size of the words, and 128 represents the size of the dense embedding vectors is taken as the output dimension.

Conv1D layer: This layer is performing one-dimensional convolution on the input sequence with 128 filters of size 5. The activation function ReLU is used in this layer and regularization of type L2 is used with a coefficient of 0.01 is applied for controlling the overfitting of the model.

MaxPooling1D layer: This layer helps in reducing the spatial size of the output we got from the previous layer by taking the maximum value in each patch of size 6.

GRU layer: This layer is used for processing the sequence in recurrent manner by taking 80 units and will returns the full sequence of outputs.

LSTM layer: This layer takes the output from the GRU layer and processes the output by taking 36 units.

Dense layer: This layer has 6 units and uses the ReLU activation function.

Output layer: This layer has 1 unit and uses the sigmoid activation function to predict binary output.

The model is compiled with the binary cross-entropy loss function, the Adam optimizer, and the accuracy metric. The model is trained with a batch size of 32 and for 25 epochs, with validation data specified for evaluating model performance.

```
from tensorflow.keras import regularizers
model = Sequential()
model.add(Embedding(5000, 128, input_length=maxlen))
model.add(Conv1D(128, 5, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.7))
model.add(MaxPooling1D(pool_size=6))
model.add(GRU(80, return_sequences=True))
model.add(LSTM(36))
model.add(Dropout(0.5))
model.add(Dense(6, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(np.array(X_train), y_train, epochs=25, batch_size=32, validation_data=(np.array(X_test), y_test))
```

The sequential neural networks is used in Natural Language Processing for Sentiment analysis, Text Classification, NER etc. For sentiment analysis, the network can be used for training the text and classifying the text based on the sentiment as positive,negative and neutral.

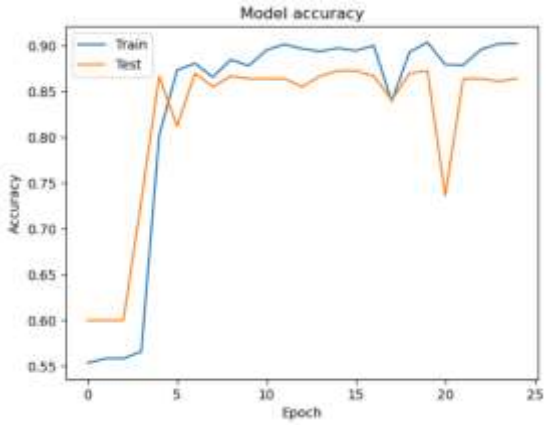
Results & Visualizations

Once the model is trained, the accuracy for the

Trained Data- 90.6%

Test Data- 86.3%

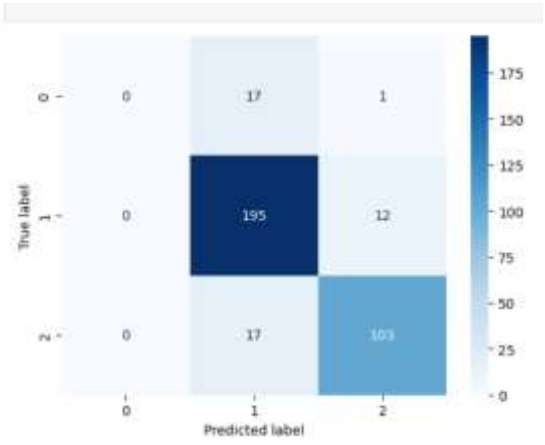
Based on the accuracy, we could conclude that the model is fit.



Evaluation Metrics for the Test data is as shown in the below image.

```
11/11 [==============================] - 0s 11ms/step
Accuracy: 86.38%
Precision: 81.98%
Recall: 86.38%
F1-score: 84.03%
```

Confusion matrix is calculated using trained and test data and is visualized using the heatmap visualization technique. 0,1,2 represent negative, neutral, positive respectively.



Limitations of the model

The limitations for this model are:

1) **Model complexity:** This model is very complex since it includes numerous parameters, and multiple layers. The model's performance on validation data must be carefully evaluated and its parameters must be adjusted to avoid overfitting.

2) **Less amount of training data:** The amount of training data improves the model's performance. If the training data is small or uneven, the model may not learn the underlying patterns in the data and may be overfit or underfit. The model can perform better by increasing the volume of the training data or by implementing data augmentation techniques.

Conclusion

The model implemented using sequential neural networks is fit and is working well for the the accuracy and the prediction is working fine for the model. The model is predicted with 10 sentences and the sentiment for the text is correctly labelled. The accuracy of the model and the other evaluation metrics are also perfectly fit.

```
it is raining
0.0
Neutral

I love going to USF games, it's so much fun!
0.9925699
Positive

the campus is peaceful in USF
0.0
Neutral

I can't wait for the semester to be over, it's been so stressful
0.31482998
Negative

USF's campus is beautiful, especially during the winter
0.94054645
Positive

I had a great time at the USF orientation, it was very informative
0.9931116
Positive

i lost my interest in joining this university
0.31482998
Negative

I'm disappointed with the quality of education at USF, it's not what I expected
0.33064747
Negative
```