Sentiment Analysis using Python

Mithun Goud Bandi
Department of Computer Science
UNC Charlotte, 28213.

Abstract

Social media is a vast platform where many people connect virtually by sharing their thoughts and emotions. People may share thoughts using comments, tweets or sharing posts, etc. The context behind the tweet can be positive, negative, or neutral. Hence, it is easy for humans to interpret the context of the tweet. However, if there are thousands of tweets it would become a hectic task for humans to read and label every tweet in the dataset. So, the Artificial Intelligence and Natural Language Processing libraries provide a framework for Sentiment Analysis by analyzing text. The main objective of this project is to focus on analyzing the sentiment of the text using NLP. Therefore, we aim to develop a sequential neural network model to predict the sentiment of the tweet.

1. Introduction

There has been a growing trend to analyze the sentiment of the text on social media because the analysis of text could help us to understand people's perceptions, or psychology, and growing trends. The application of sentiment analysis ranges from analyzing people's perceptions of a public company or electoral debates. This analysis could guide a company to develop new products, or the Presidential candidate could design new strategies according to the analysis. In this way, text analysis has become a key such that many techniques have been developed over the years to analyze the sentiment of the text.

Sentiment analysis is a field of natural language processing that deals with the identification, extraction, and analysis of subjective information from text data. It involves using computational methods to automatically identify and classify the sentiment expressed in a piece of text, such as positive, negative, or neutral.

In this project, we are going to develop a sequential neural network model that trains and classifies the sentiment of the text. Usually, these models are trained on large datasets and are capable of learning the underlying patterns and relationships between the words in the text data, which makes them well-suited for this task. A sequential neural network is a type of neural network architecture in which layers are layers organized in a sequential sequence, with the output of one layer acting as the input for the subsequent layer. The first layer receives the input data and processes it before passing the results to the second layer, and so on until the final output is generated.

A sequential neural network in NLP, which includes interpreting the meaning of the words and their relationships within the phrase, may take a series of words as input and predict the sentiment of the text. In order to estimate the sentiment of the input text, the model learns to extract key aspects from the text.

2. Data Collection

The data collection involves setting up a Twitter developer account and accessing the tweets using the Tweepy library in Python. The data collection involves a series of steps:

- **Setting up a Twitter developer account:** The Twitter app provides a framework to extract tweets using a Twitter developer platform. In order to extract the tweets, we need to sign up for a Twitter developer account. The next step involves setting up an app in the developer platform.

- **Create an app in the Twitter developer platform:** This step involves creating a standalone app in the project that helps us read and write tweets. This app can be used to authenticate the requests to the Twitter API.

- **API and access tokens:** Generate API token, API secret token, access token, and access secret token. Store the keys in order to authenticate with the Twitter API using the tweepy library framework.

- **Use the tweepy library:** Python provides a tweepy framework handy to authenticate with the Twitter API. Most of the read and write functions to extract tweets are already in-built into the library. Hence, use the pre-defined in-built functions to authenticate with the Twitter API and extract tweets.

- **Extracting the tweets:** Extract the tweets of the university using hashtags and extract the information such as username, tweet text, creation date, and time of the tweet and append them to a list.

- **Store the tweets in a data frame:** The last part of the step includes storing the tweets into a data frame using the '.csv' or 'xlsx' extension.

3. Data preprocessing

Data preprocessing is a crucial step that involves cleaning the data and transforming the raw data such that the data is suitable for training the model. In sentiment analysis, cleaning data includes removing punctuation, and stop words. However, the tweets could also possess hashtags, mentions, and URLs in them. Hence, that data must be cleaned by removing the hashtags, mentions, and URLs. The detailed process of data preprocessing is mentioned below:

- **Converting the text to lowercase:** The entire tweet text is converted into lowercase using the lower( ) function which is defined in the Python libraries.

- **Remove URLs from the tweets:** Some tweets may consist of URL links embedded in the text. Hence, if any tweets contain hyperlink it needs to be cleaned. Therefore, import regular expressions from the Python library and create a regular expression that needs to remove any URL link that is embedded within the tweet text. The regular expression **‘*http*\*S*+’** can remove any link that is written within the text.

- **Remove hashtags and mentions from the tweets:** According to the trend, people include hashtags and mention the people in the tweets. Therefore, to remove the hashtags and mentions from the tweets use the regular expression **‘@\*S*+|#\S+’.** This regular expression facilitates in removing the hashtags and mentions from the tweet.

- **Remove punctuations from the text:** The text consists of punctuations such as commas (,), question marks (?), etc. These punctuations can be removed from the tweets using the regular expression **‘*[*^\*w*\s]’**.

- **Tokenize the text:** Tokenize the text into individual words using the NLTK library to import word_tokenize function. This helps to tokenize the text into individual words such that we can remove stop words and lemmatize the text.

- **Remove stop words:** The stop words can be removed from the text using the NLTK framework by importing the stopwords. Now, identify the stopwords using the framework provided by NLTK and remove the stopwords from the text.

- **Lemmatization:** Lemmatization is a natural language processing technique that involves reducing words to their base form known as the lemma. This step increases the accuracy of text analysis by reducing the complexity of the vocabulary. The lemmatization can be done simply by importing WordNetLemmatizer( ) from the NLTK. This helps to lemmatize the text.

- **Join the tokens back to a single string:** Now, join the tokens back into a single string to perform text analysis.

- **Check for null values in the data:** Make sure to check the null values in the dataset. The null values in the dataset can have a significant impact on the accuracy or performance of the model. It could also result in bias in the text analysis leading to inaccurate predictions and poor model performance. Therefore, it is important to check for null values and drop the null values from the dataset.

These preprocessing steps facilitate converting the raw data into a suitable format that is suitable for training the model.

4. Feature extraction

The feature extraction involves creating a matrix of word counts for each tweet and using TF-IDF to weigh the importance of each word. The following steps are involved in feature extraction:

- **Create a bag of words:** A bag of words is the representation that describes the count of words within a document. So, we need to tokenize the preprocessed text to create a bag of words. This can be done by importing the CountVectorizer( ) from the NLTK library such that it facilitates creating and storing the bag of words.

- **Weigh the importance of each word:** Using TF-IDF weigh the importance of each word. The TF-IDF score for a word is higher if the word frequently appears in a tweet.

5. Model architecture

The sequential neural network model is developed using TensorFlow. The model incorporates several layers. Before creating the model that dataset is split into 80% of the training dataset and 20% of the testing dataset using the train_test_split method imported from sklearn library. Now, tokenize the text and pad the sequences of the text to the max length of sequences. The model consists of layers:

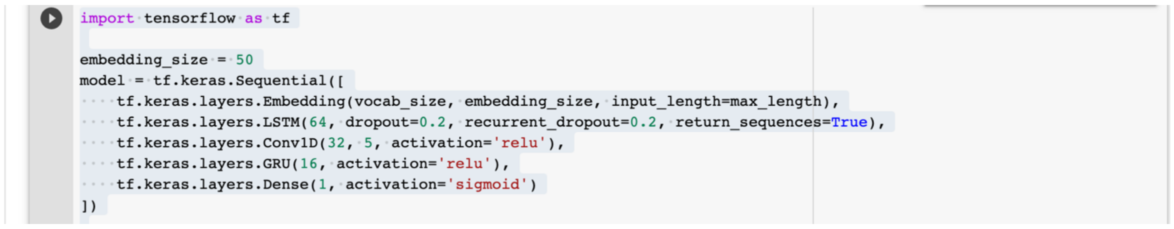
- **Embedding layer:** The first layer is the embedding layer which takes the integer encoded input and maps each integer to a dense vector. Hence, similar words have familiar vector representations.

- **LSTM layer:** The second layer of the model is incorporated with the Long short-term memory (LSTM) layer. It is a type of recurrent neural network layer that is capable of long-term dependencies. These are used for sequence prediction problems which can overcome the vanishing gradient issue.

- **Convolutional layer:** The third layer of the model is the convolutional layer which can extract sequences of n consecutive words.

- **GRU layer:** The fourth layer of the model is the Gated recurrent unit (GRU) layer which is similar to the LSTM because it is also a type of recurrent neural network layer.

- **Dense layer:** This is the final output layer which applies a linear transformation to the output of the previous GRU layer.



```
import tensorflow as tf

embedding_size = 50
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_size, input_length=max_length),
    tf.keras.layers.LSTM(64, dropout=0.2, recurrent_dropout=0.2, return_sequences=True),
    tf.keras.layers.Conv1D(32, 5, activation='relu'),
    tf.keras.layers.GRU(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Figure 1: Sequential neural network

The sequential neural network model is developed using TensorFlow and incorporates five layers to efficiently process and analyze textual data for sentiment analysis. The architecture starts with an embedding layer that transforms input sequences of word indices, padded to the max length of a sequence in the list of tokenized sequences, into dense vector representations of embedding size. These word embeddings help the model capture semantic relationships between words in the text. Following the Embedding layer, a Long Short-Term Memory (LSTM) layer with 64 hidden units processes the sequence of word vectors, considering long-range dependencies in the text. To prevent overfitting, the LSTM layer employs a dropout rate of 0.2 The model then utilizes a 1D convolutional layer with 32 filters and a kernel size of 5, which applies filters of size 5 to the output of the sequence by the LSTM layer. This step aids in the extraction of meaningful local features from the data. The model now proceeds with a Gated Recurrent Unit (GRU) layer containing 16 hidden units and ReLU activation. The GRU layer refines the learned features by capturing additional temporal relationships in the data. Lastly, a Dense layer with a single output with a sigmoid activation function is used to output the predicted probability of the positive class. This versatile model architecture, combining the strengths of various layers such as LSTM, Conv1D, and GRU, enables the effective handling and analysis of complex text data for classification tasks, including sentiment analysis. The model has a total of 264,379 trainable parameters, providing substantial learning capacity for accurate sentiment prediction.

6. Results

The model was trained and evaluated on a dataset that was split into 80% for training and 20% for testing. Over the course of 5 epochs, the model achieved a training accuracy of 97.45% and a testing accuracy of 72.77%. The training and validation loss and accuracy plots show that the model performed well on the training data, with a decreasing loss and increasing accuracy over time. However, the validation loss and accuracy exhibited fluctuations, indicating that the model may not generalize as well to new data.

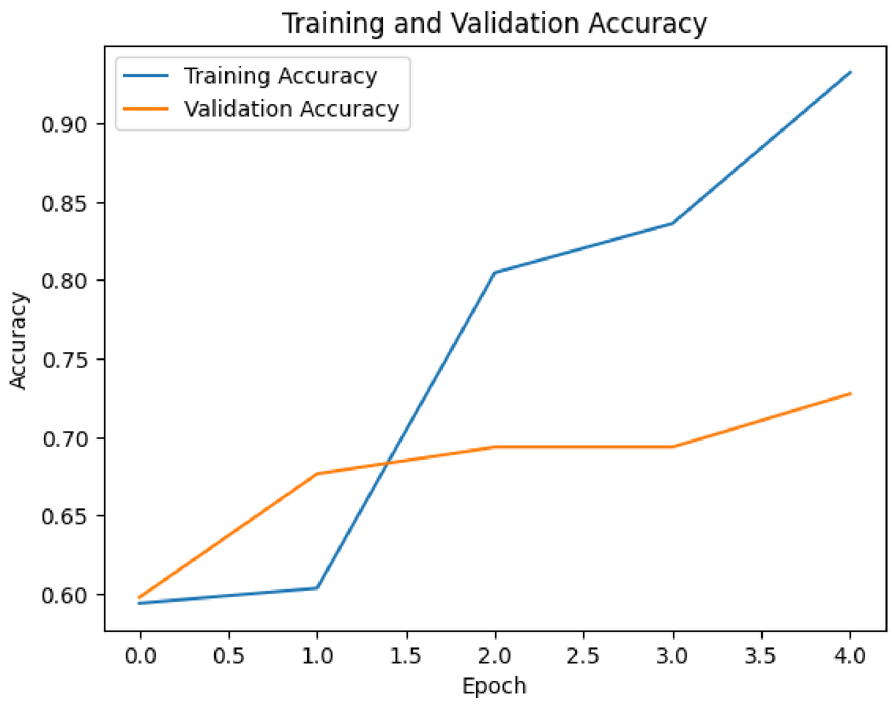


Figure 2: Training and Validation accuracy

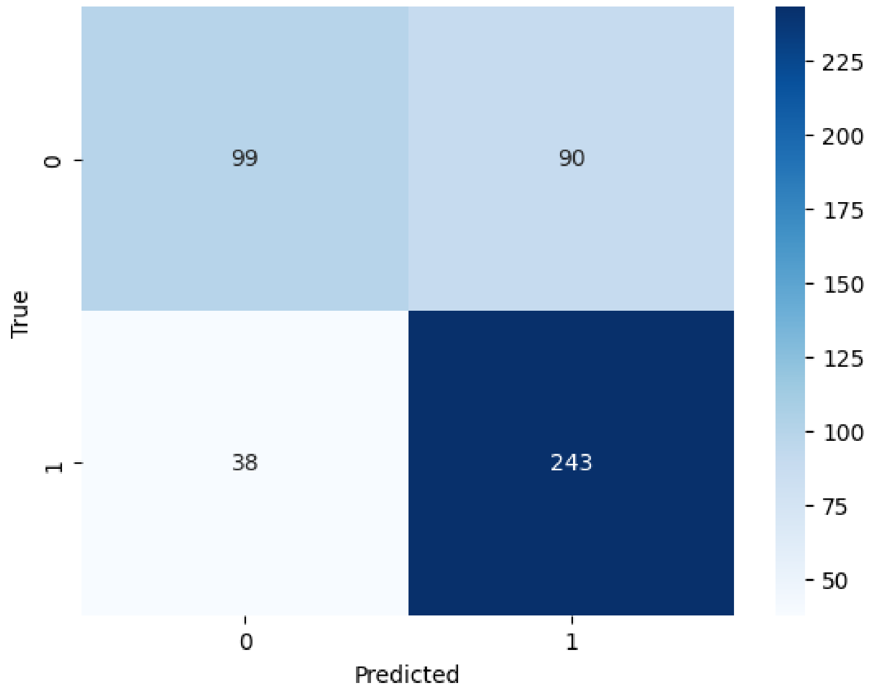


Figure 4: Confusion matrix

The model appears to have achieved a reasonable level of accuracy, precision, recall, and F1-score, with an overall accuracy of 73% and a weighted average F1-score of 0.72. However, the model's performance can be further improved, particularly in terms of its generalization to new data. The model's precision and recall were higher for the positive class compared to the negative class, indicating potential room for improvement in detecting negative sentiments. Future work could involve exploring alternative model architectures, further data preprocessing techniques, and hyperparameter tuning to improve the model's performance.

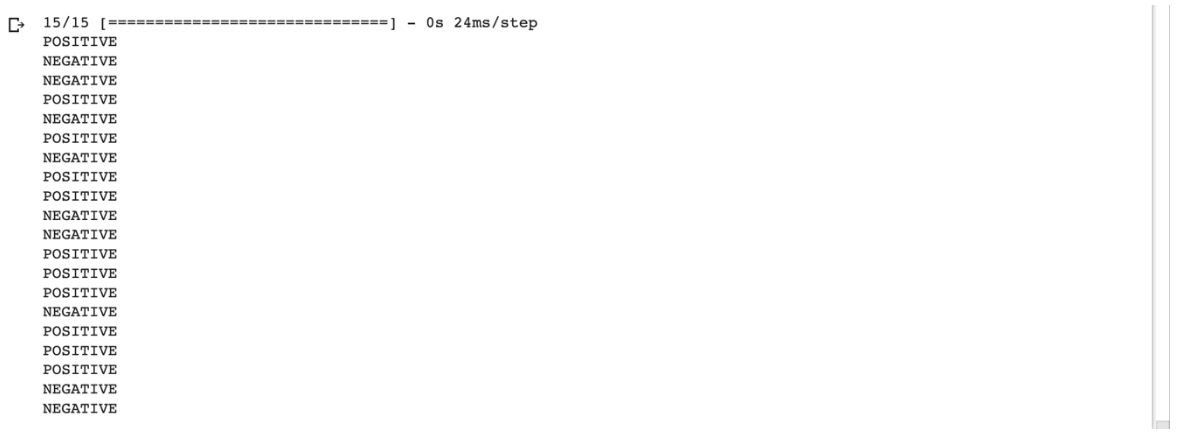
7. Limitations of the model

The limitations of the model include the following:

- **Class imbalance issues:** The model may face inconsistencies due to class imbalance, which can affect its performance in predicting the sentiment of the text.
- **Capturing dependencies:** While LSTMs can capture short-term dependencies in sequential data, they may struggle with long-term dependencies.
- **Small dataset:** The Twitter API allows only 1500 tweets to be read every month. Hence, a small dataset is used for training the model. Therefore, this can limit the model's ability to accurately classify sentiments on new data.

8. Conclusion

Sentiment analysis is an important application of natural language processing, which has numerous real-world applications. In this project, we used a sequential neural network model to perform sentiment analysis on a Twitter dataset using a sequential neural network model. The text data was first preprocessed, features were retrieved using the bag-of-words method and TF-IDF, and our model was trained using binary cross-entropy loss and the Adam optimizer. Our model has a good accuracy score, but there is still potential for development. We also noted the model's shortcomings, such as the requirement for larger amounts of data. Hence, we can conclude that the model created can successfully classify the sentiment of the text.



```
15/15 [==============================] - 0s 24ms/step
POSITIVE
NEGATIVE
NEGATIVE
POSITIVE
NEGATIVE
POSITIVE
NEGATIVE
POSITIVE
POSITIVE
NEGATIVE
NEGATIVE
POSITIVE
POSITIVE
POSITIVE
NEGATIVE
POSITIVE
POSITIVE
POSITIVE
NEGATIVE
NEGATIVE
```

Figure 4: Predictions of the model