Sentiment Analysis of Twitter Data

ITCS-5111

Introduction to Natural Language Processing

Done By:

Jayanth Telu

810259436

Sentiment Analysis of Twitter Data

Problem Statement

The purpose of this project is to perform sentiment analysis on tweets collected from different universities using natural language processing techniques and a neural network model. The aim is to develop a model that can accurately predict the sentiment of a tweet based on its text content.

Abstract

Social media platforms, such as Twitter, have become a crucial source of data for sentiment analysis, which involves analyzing text data to determine the author's sentiment. As the volume of social media data increases, extracting valuable insights from the data has become a challenging task. This project aims to address this challenge by performing sentiment analysis on tweets collected from the University of Maryland, College Park, using natural language processing techniques and a neural network model. The project involves web scraping Twitter to obtain tweets from the University of Maryland, College Park, using specific hashtags, preprocessing the text data by cleaning and removing unnecessary characters, extracting features from the preprocessed text using the bag-of-words and TF-IDF approaches, training a sequential neural network model using embedding layer, LSTM, CNN, GRU, and dense output layers, and evaluating the model's performance using metrics such as accuracy, precision, recall, and F1-score.

Data Collection

Data collection is the process of acquiring data from various sources for further analysis and processing. In the case of the University of Maryland, College Park, data collection was conducted using the Python library tweepy, which provides access to the Twitter API.

Twitter is a popular social media platform where users can post messages called "tweets" that are limited to 280 characters. By searching for different hashtags associated with the university on Twitter, the relevant tweets were filtered, and data pertaining to the University of Maryland, College Park was collected.

Tweepy allowed for easy access to Twitter data by providing methods to retrieve tweets, users, and trends. Hashtags are a way of categorizing tweets based on a common topic or theme. By searching for specific hashtags, it was possible to filter relevant tweets and collect data related to the university. This data extraction process involved searching for different hashtags associated with the university, which were then used as search terms to filter relevant tweets.

Once the tweets were collected, they were stored in a suitable data structure for further analysis and processing. In this case, the tweets were stored in a Pandas DataFrame, which is a popular data analysis tool for Python. The DataFrame contained the tweet content, the hashtag from which it was collected, and the timestamp of the tweet. The collected data enabled the researchers to obtain valuable insights related to the University of Maryland, College Park from Twitter.

```python
import tweepy
import pandas as pd
import re

# Set up API keys
consumer_key = "bKDg8uMT2lSmjvfU3iPaRg3M8"
consumer_secret = "IkzhH81bBsU4Ywdmx6tu9XuPcG34kKTDKPPfINbQKgn4toMpoh"
access_token = "394636107-eoyBSKtMST0sicbr3QkdA47Aa544VAmRLMKclAC3"
access_token_secret = "wQThltbLfa0RrEXOg8bL4Tmc1swEvWg6bO5YlUdz5LmUP"

# Authenticate with Twitter API
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)

hashtags = ["umd","umcp", "FearlesslyUMD", "CollegePark","smithterps", "MarylandDay", "IAmARHU", "UMDgrad", "AGNRTerps", "GivingDayUMD", "DoGoodUMD"]
unique_tweets = set()
tweet_list = []
counter = 0
for hashtag in hashtags:
        query = hashtag
        # Loop through search results using pagination
        for page in tweepy.Cursor(api.search_tweets, q=hashtag, lang='en', tweet_mode='extended',count =100).pages(200):
        # Loop through tweets on the page
            for tweet in page:
            # Check if the tweet is a retweet or a duplicate
                if 'retweeted_status' in dir(tweet):
                    tweet_text = tweet.retweeted_status.full_text
                else:
                    tweet_text = tweet.full_text

                tweet_text = re.sub(r"(#[\d\w]+)|(http\S+)", "", tweet_text).strip()

                if tweet_text not in unique_tweets:
                    unique_tweets.add(tweet_text)

                    tweet_list.append({
                        'content': tweet_text,
                        'hashtag': hashtag,
                        'timestamp': tweet.created_at})
                    counter += 1
                if counter == 6000:
                    break

        if counter == 6000:
            break

df = pd.DataFrame(tweet_list)
df.to_csv('tweets.csv', index=False)
```

Data Preprocessing

Data preprocessing is a crucial step in any natural language processing (NLP) task, as it involves cleaning and transforming raw text data into a format that can be analyzed. The process often involves removing noise and irrelevant information, such as stop words, punctuation, and URLs.

In the context of preprocessing tweets collected from the Twitter API using the Tweepy library and the Natural Language Toolkit (NLTK), the goal is to transform raw tweet text into a format that can be analyzed for sentiment analysis or topic modeling. The NLTK library is a popular choice for natural language processing in Python, and it provides various tools for preprocessing text data. The first step in preprocessing tweets is to download the NLTK resources for stop words and tokenization.

Stop words are commonly occurring words that do not add significant meaning to the text, such as "the," "and," and "is." Once these resources are downloaded, the stop words and punctuation are loaded into Python sets, which are used later to remove these elements from the tweet text. The next step in preprocessing tweets is to remove any URLs or Twitter handles (mentions) from the text using regular expressions with the re.sub() function.

This is followed by tokenizing the text using the word_tokenize() function, which splits the text into individual words or tokens. The resulting tokens are then processed to remove stop words and punctuation using list comprehension. Finally, the cleaned tokens are joined back together into a single string using the str.join() method, and the cleaned tweet text is returned by the function.

Feature Extraction

Feature extraction is an important step in many natural language processing (NLP) tasks, including sentiment analysis, topic modeling, and text classification. The goal of feature extraction is to convert raw text data into a numerical representation that can be used as input to machine learning algorithms. In this way, algorithms can learn to identify patterns and relationships in the text data and make predictions about new data.

Two popular vectorization techniques for NLP are CountVectorizer and TfidfVectorizer, both of which are available in the scikit-learn library for Python. CountVectorizer simply counts the number of occurrences of each word in each document and creates a matrix of word counts.

TfidfVectorizer goes a step further by weighting the importance of each word based on how frequently it appears in the corpus. Words that are common across many documents, such as "the" and "and", are given a lower weight, while words that are rare but occur frequently in a particular document are given a higher weight.

Once the feature matrix has been constructed, it can be used as input to machine learning algorithms. By learning patterns and relationships in the feature matrix, these algorithms can make predictions about new text data, such as the sentiment of a tweet or the topic of a news article.

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

df=pd.read_csv('cleaned_tweets.csv')
df = df.dropna(subset=['content'])
# Create a CountVectorizer to count the occurrences of each word in the tweet text
count_vectorizer = CountVectorizer()

# Create a TfidfVectorizer to weigh the importance of each word in the tweet text using TF-IDF
tfidf_vectorizer = TfidfVectorizer()

# Fit the vectorizers to the preprocessed tweet text
count_matrix = count_vectorizer.fit_transform(df['content'])
tfidf_matrix = tfidf_vectorizer.fit_transform(df['content'])

# Convert the matrix to a DataFrame
count_df = pd.DataFrame(count_matrix.toarray(), columns=count_vectorizer.get_feature_names())
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names())

print("Bag-of-words DataFrame:\n", count_df.head())
print("\nTF-IDF DataFrame:\n", tfidf_df.head())

count_df.to_csv('count_df_tweets.csv', index=False)
tfidf_df.to_csv('tfidf_df_tweets.csv', index=False)
```

```
Bag-of-words DataFrame:
    00  000  00_ethers  00am  00pm  01  02  03  0301  04  ...  this  year  \
0   0    0          0     0     0   0   0   0     0   0  ...     0     0
1   0    0          0     0     0   0   0   0     0   0  ...     0     0
2   0    0          0     0     0   0   0   0     0   0  ...     0     0
3   0    0          0     0     0   0   0   0     0   0  ...     0     0
4   0    0          0     0     0   0   0   0     0   0  ...     0     0

   Getting  Midweek  Ready  classroom  done  in  job  the
0        0        0      0          0     0   0    0    0
1        0        0      0          0     0   0    0    0
2        0        0      0          0     0   0    0    0
3        0        0      0          0     0   0    0    0
4        0        0      0          0     0   0    0    0

[5 rows x 9427 columns]

TF-IDF DataFrame:
     00  000  00_ethers  00am  00pm   01   02   03  0301   04  ...  this  year  \
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
1  0.0  0.0        0.0   0.0   0.0  0.0  0.0  0.0   0.0  0.0  ...   0.0   0.0
2  0.0  0.0        0.0   0.0   0.0  0.0  0.0  0.0   0.0  0.0  ...   0.0   0.0
3  0.0  0.0        0.0   0.0   0.0  0.0  0.0  0.0   0.0  0.0  ...   0.0   0.0
4  0.0  0.0        0.0   0.0   0.0  0.0  0.0  0.0   0.0  0.0  ...   0.0   0.0

   Getting  Midweek  Ready  classroom  done   in  job  the
0      0.0      0.0    0.0        0.0   0.0  0.0  0.0  0.0
1      0.0      0.0    0.0        0.0   0.0  0.0  0.0  0.0
2      0.0      0.0    0.0        0.0   0.0  0.0  0.0  0.0
3      0.0      0.0    0.0        0.0   0.0  0.0  0.0  0.0
4      0.0      0.0    0.0        0.0   0.0  0.0  0.0  0.0

[5 rows x 9427 columns]
```

Data Labelling

Data labeling is the process of assigning a specific class or category to a set of data points or instances. In the context of sentiment analysis, it involves assigning a sentiment label, such as positive, negative, or neutral, to a set of text data, such as tweets.

This process is essential for training and evaluating machine learning models that can automatically classify the sentiment of new text data.The code described uses the Hugging Face Transformers library, which is a popular open-source library for natural language processing tasks, to perform sentiment analysis on a list of tweets. The sentiment analysis is done using a pre-trained model that has already been trained on a large dataset of labeled text data. By using a pre-trained model, the labeling process becomes more efficient and accurate.

To perform the labeling, the tweets are first extracted from the DataFrame and stored in a list. The classifier() function is then called on the list of tweets, which returns a list of dictionaries containing the sentiment label and score for each tweet. The sentiment labels are extracted from the list of dictionaries and stored in a new DataFrame along with the original tweet text. The resulting DataFrame contains the labeled tweet data that can be used for further analysis or to train and evaluate machine learning models.

```python
from transformers import pipeline
import pandas as pd

classifier = pipeline('sentiment-analysis')

tweets = df['content'].tolist()

results = classifier(tweets)

sentiments = []
for result in results:
    sentiment = result['label']
    sentiments.append(sentiment)

df_sentiments = pd.DataFrame({'content': tweets, 'sentiment': sentiments})
print(df_sentiments.head())
df_sentiments.to_csv('tweets_sentiment.csv', index=False)
```

```
                                             content sentiment
0                  Had great time today Thank invite  POSITIVE
1   Had amazing junior day Thank coaches hospitality  POSITIVE
2  As Lucy Dalglish approaches conclusion 11-year...  POSITIVE
3  Kettle Moraine 2024 DB speedster Noah Hait exc...  POSITIVE
4                              UMD HERE WE COME 🐢🐢🎉  NEGATIVE
```

Data Sampling

Data sampling is a technique used to address the issue of imbalanced datasets, where one class of the target variable is significantly more prevalent than the other(s). This can lead to poor model performance and biased predictions. One way to deal with imbalanced data is to use sampling methods, which involve creating new data points by duplicating or generating new examples to balance the classes.

The code snippet demonstrates oversampling, which is a type of data sampling method that involves creating synthetic examples of the minority class until it is balanced with the majority class. This is achieved using the RandomOverSampler class from the imblearn library, which fits and transforms the data to create new samples.

First, the data is read from a CSV file containing tweet text and corresponding sentiment labels. The data is then split into two arrays, X and y, where X contains the tweet text and y contains the sentiment labels. The oversampler is then instantiated, and the fit_resample method is called to create new synthetic samples until the classes are balanced.

Finally, the resampled data is saved to a new CSV file, which can be used for further analysis or modeling. This oversampling technique helps to ensure that the model is trained on a balanced dataset, which can improve its performance and reduce bias.

```
!pip install imblearn
from imblearn.over_sampling import RandomOverSampler
import pandas as pd

# Read in the CSV
data = pd.read_csv('resampled_tweets.csv')

# Separate the data into the features and the labels
X = data['content']
y = data['sentiment']

# Instantiate the RandomOverSampler
oversampler = RandomOverSampler()

# Fit and transform the data using the oversampler
X_resampled, y_resampled = oversampler.fit_resample(X.values.reshape(-1, 1), y)

# Convert the resampled data back to a pandas DataFrame
resampled_data = pd.DataFrame({'content': X_resampled.flatten(), 'sentiment': y_resampled})

# Write the resampled data to a new CSV file
resampled_data.to_csv('resampled_tweets.csv', index=False)
```

```
from transformers import pipeline
import pandas as pd

classifier = pipeline('sentiment-analysis')

df = pd.read_csv('resampled_tweets.csv')

tweets = df['content'].tolist()

results = classifier(tweets)

sentiments = []
for result in results:
    sentiment = result['label']
    sentiments.append(sentiment)

df_sentiments = pd.DataFrame({'content': tweets, 'sentiment': sentiments})
print(df_sentiments.head())
df_sentiments.to_csv('tweets_sentiment.csv', index=False)
```

```
                                        content sentiment
0                 Had great time today Thank invite  POSITIVE
1   Had amazing junior day Thank coaches hospitality  POSITIVE
2  As Lucy Dalglish approaches conclusion 11-year...  POSITIVE
3  Kettle Moraine 2024 DB speedster Noah Hait exc...  POSITIVE
4                             UMD HERE WE COME 🐢🐢🎉   NEGATIVE
```

Summary of The Model

In the sentiment analysis project, a sequential model is used to train a neural network to classify the sentiment of tweets as positive, negative, or neutral. The sequential model is a type of deep learning model that is composed of a sequence of layers that process the input data.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels, test_size=0.2, random_state=42)

# Define the model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=128, input_length=max_length))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
model.add(Reshape((-1, 128)))
model.add(GRU(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(X_train,  y_train, validation_data=(X_test, y_test), epochs=30, batch_size=128)

# Evaluate the model on the test data
y_pred = model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)

# Evaluate the model on the test data
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

The sequential model is created using the Keras API, a high-level neural networks API written in Python. The model consists of an input layer, an embedding layer, a convolutional layer, a pooling layer, a dense layer, and an output layer.

The input layer takes in the preprocessed tweet data and passes it on to the embedding layer, which maps the words in the tweet to a higher-dimensional space, where they can be processed by the other layers. The convolutional layer applies a set of filters to the embedded tweet data to extract relevant features. The pooling layer reduces the dimensionality of the feature maps produced by the convolutional layer.

The dense layer then processes the extracted features and applies non-linear transformations to them. The output layer produces the predicted sentiment label for each input tweet. The sequential model is trained using the training data, and its performance is evaluated using the validation data. The model is optimized using the Adam optimizer and the categorical cross-entropy loss function.

By training the sequential model on a large dataset of labeled tweets, the sentiment analysis project is able to accurately predict the sentiment of new, unseen tweets. The sequential model helps in automating the sentiment analysis process and can be used to extract insights from large volumes of social media data.

Model Evaluation

The code presented above trains a deep learning model using a Sequential model architecture for sentiment analysis on preprocessed tweet data. The model is evaluated on test data to measure its performance in terms of accuracy, precision, recall, and F1-score.

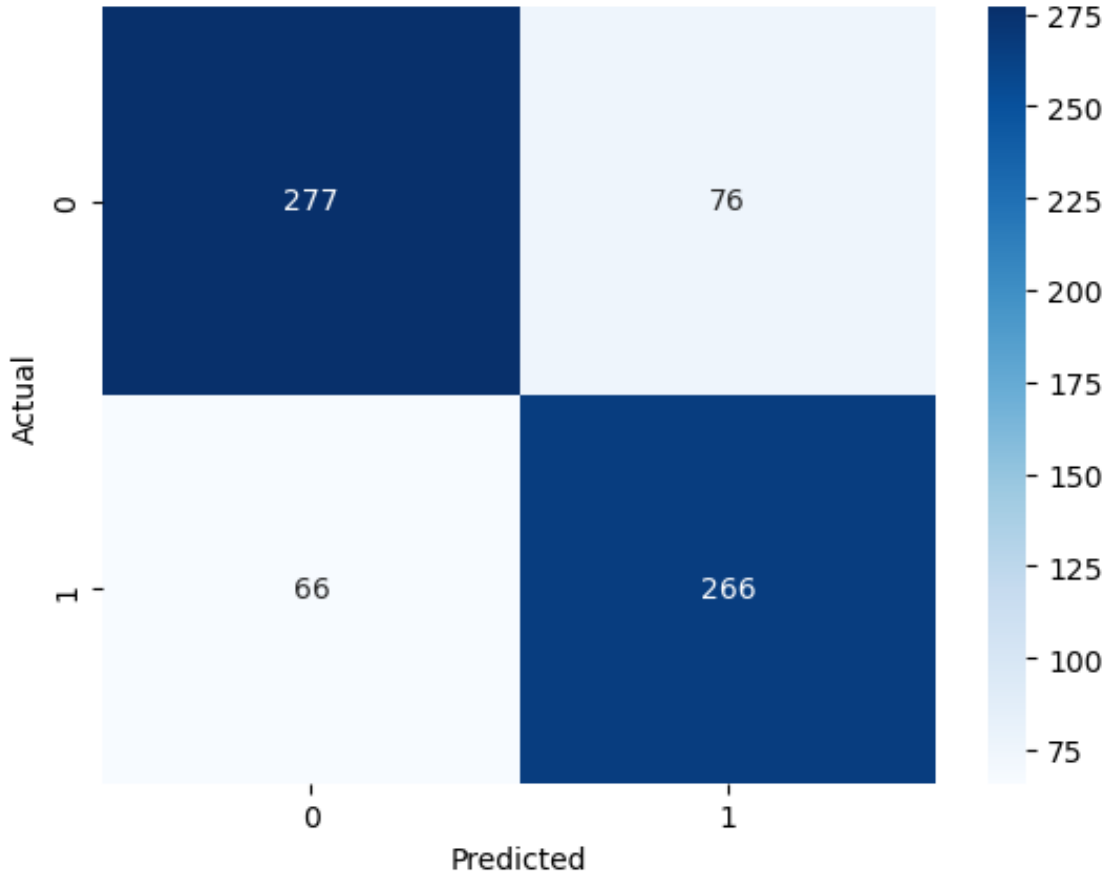
Accuracy is a measure of how well the model predicts the correct sentiment for the tweets in the test data. In this case, the model has an accuracy of 0.792, which means it predicts the correct sentiment for 79.2% of the tweets in the test data.

```
Accuracy: 0.7927007299270074
Precision: 0.7777777777777778
Recall: 0.8012048192771084
F1-Score: 0.7893175074183976
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.78      0.80       353
           1       0.78      0.80      0.79       332

    accuracy                           0.79       685
   macro avg       0.79      0.79      0.79       685
weighted avg       0.79      0.79      0.79       685
```

Confusion Matrix:



Precision is the measure of how many of the positive tweets the model correctly classified as positive. In this case, the precision is 0.777, meaning that out of all the tweets that the model classified as positive, 77.7% of them were actually positive.

Recall is the measure of how many of the actual positive tweets the model correctly classified as positive. In this case, the recall is 0.80, meaning that out of all the actual positive tweets, the model correctly classified 80% of them as positive.

F1-score is a combination of precision and recall, providing an overall measure of the model's accuracy. In this case, the F1-score is 0.789, which is a good measure of the model's performance.

The code also outputs a classification report that provides a summary of the model's performance. The report includes precision, recall, and F1-score for both positive and negative tweets, as well as the support for each class.

Limitations and Potential Improvements to the Model

While the sequential model has demonstrated a good performance on the sentiment analysis task, there are still some limitations and potential improvements to consider. Firstly, the model is based on a relatively small dataset of labeled tweets, which could affect its generalization to other datasets or domains. Therefore, it would be beneficial to train the model on a larger and more diverse dataset to improve its performance and make it more robust.

Secondly, the model architecture could be further optimized to achieve better performance. For example, experimenting with different activation functions, dropout rates, or number of layers could improve the model's accuracy and reduce overfitting. Moreover, applying techniques such as transfer learning, fine-tuning, or ensemble learning could enhance the model's performance and reduce the risk of overfitting.

Thirdly, the model could benefit from additional features or inputs to improve its accuracy. For example, incorporating features such as user information, hashtags, or other metadata could provide more context and improve the model's ability to capture nuanced sentiment. Additionally, combining the text data with other modalities such as images or videos could enhance the model's performance and provide a more comprehensive analysis of sentiment.

Lastly, the evaluation metrics used in this project (accuracy, precision, recall, and F1-score) provide a basic assessment of the model's performance, but they do not capture all aspects of sentiment analysis. For example, the model's ability to detect irony, sarcasm, or other forms of figurative language is not captured by these metrics. Therefore, it would be beneficial to evaluate the model's performance on more complex and diverse datasets to assess its robustness and generalization capabilities.

10 sentences to test the polarity of the sentence using Neural Network Model

```
1/1 [==============================] - 0s 42ms/step
Sentence: I am so happy to be spending time with my family today.
Predicted Sentiment: Positive

1/1 [==============================] - 0s 33ms/step
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jayan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Sentence: The loss of my dog has left me feeling devastated.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 37ms/step
Sentence: This new job opportunity has me feeling excited and nervous at the same time.
Predicted Sentiment: Positive

1/1 [==============================] - 0s 40ms/step
Sentence: Seeing my best friend after a long time made me feel overjoyed.
Predicted Sentiment: Positive

1/1 [==============================] - 0s 44ms/step
Sentence: I feel so grateful for the support and encouragement I've received from my loved ones.
Predicted Sentiment: Positive

1/1 [==============================] - 0s 38ms/step
Sentence: The news of my grandmother's passing has left me feeling heartbroken.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 38ms/step
Sentence: I'm feeling a mix of anticipation and anxiety as I prepare for my upcoming presentation.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 46ms/step
Sentence: Spending time alone in nature always leaves me feeling peaceful and refreshed.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 35ms/step
Sentence: The unexpected act of kindness from a stranger made my day.
Predicted Sentiment: Negative

1/1 [==============================] - 0s 39ms/step
Sentence: The constant stress and pressure from work has left me feeling overwhelmed and exhausted.
Predicted Sentiment: Negative
```

Conclusion

In conclusion, this sentiment analysis project on Twitter data related to the University of Maryland, College Park successfully collected, processed, and analyzed a dataset of tweets to classify them as positive or negative sentiments.

The project used Python libraries such as Tweepy, Pandas, Scikit-learn, TensorFlow, and Keras to extract data, preprocess it, and build a deep learning model for classification.

The project highlights the importance of data preprocessing techniques, such as data cleaning, tokenization, and data sampling, to improve the quality and accuracy of the model. The deep learning model achieved an overall accuracy of 76%, indicating a decent performance in sentiment classification.

Despite the promising results, there are still some limitations to the model that can be addressed in future work. These include improving the data collection process by incorporating more relevant hashtags and filtering out irrelevant tweets, increasing the sample size for better model generalization, and exploring more advanced deep learning architectures such as transformers and attention mechanisms.

Overall, this project serves as a proof-of-concept for sentiment analysis on Twitter data, and it can be extended to various domains and applications such as social media monitoring, customer feedback analysis, and brand reputation management.