Predicting Sentiment from Tweets

Introduction

Sentiment Analysis is a branch of Natural Language Processing (NLP) that involves identifying and extracting subjective information from text. It is widely used in various applications such as social media monitoring, customer feedback analysis, and market research. The main goal of this project is to build a sentiment analysis model that classifies Twitter tweets as positive or negative.

In this project, I, Albin Ajeti, leveraged a Twitter dataset containing labeled tweets, which are pre-tagged as "positive" or "negative." I employed several machine learning models to determine which algorithm performs best at classifying the sentiments of these tweets. The dataset used is from the NLTK library, specifically the twitter_samples corpus, which contains a collection of positive and negative tweets.

Data Collection

For this project, I used a dataset from the Natural Language Toolkit (NLTK) library, which contains a collection of tweets classified by sentiment (positive and negative). The NLTK library provides convenient access to datasets, and we used the 'twitter_samples' corpus for this sentiment analysis task.

Once NLTK was set up, I imported the essential libraries needed for data cleaning, text processing, and splitting the dataset into training and testing subsets. The following libraries were used in the process:

```
import nltk
import string
import emoji
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, LancasterStemmer, WordNetLemmatizer
import pandas as pd
from sklearn.model_selection import train_test_split
```

These libraries are fundamental for performing various tasks such as removing stopwords, tokenizing the text, stemming or lemmatizing words, and splitting the dataset.

Downloading the Dataset

The dataset used for this project is available through the twitter_samples corpus in NLTK. This dataset includes a collection of tweets that are labeled as either positive or negative, which are crucial for performing sentiment analysis.

To download the necessary dataset and resources, I used the following code:

```
# Ensure necessary downloads
nltk.download('twitter_samples')
```

This command downloads the twitter_samples dataset, which contains the tweets we used for sentiment analysis.

Setting Path for NLTK Data

In some cases, NLTK needs to know where to store and retrieve its datasets and resources. To ensure smooth access to these resources, I manually set the path for NLTK data:

```
import os
# Set nltk data path manually
nltk_data_path = os.path.join(os.getcwd(), 'nltk_data')
if not os.path.exists(nltk_data_path):
    os.makedirs(nltk_data_path)
nltk.data.path.append(nltk_data_path)
```

Downloading Additional NLTK Resources

In addition to the ‘twitter_samples’ dataset, I also downloaded other essential NLTK resources, including stopwords and tokenization tools. These resources are required for processing the text data and preparing it for analysis. The following downloads were executed:

```
# Ensure required downloads
nltk.download('twitter_samples', download_dir=nltk_data_path)
nltk.download('stopwords', download_dir=nltk_data_path)
nltk.download('punkt', download_dir=nltk_data_path)
nltk.download('punkt_tab', download_dir=nltk_data_path)
```

The stopwords resource contains a list of common words (such as "the," "a," and "in") that are typically removed during text processing. The punkt resource provides tokenization models, which are used to split the text into words or sentences.

By specifying the download directory (download_dir=nltk_data_path), I ensured that the files were stored in the correct location for later use.

Data Preparation and Organization

Once the necessary NLTK datasets were downloaded, I proceeded to extract the positive and negative tweets from the twitter_samples corpus. The NLTK library offers an easy way to access different components of the dataset. Specifically, I used the strings() function to load the positive and negative tweets from the positive_tweets.json and negative_tweets.json files, respectively.

```
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
print('Number of positive tweets:', len(all_positive_tweets))
print('Number of negative tweets:', len(all_negative_tweets))
```

Organizing data into a DataFrame

To simplify further manipulation and analysis of the dataset, I decided to combine the positive and negative tweets into a single **pandas DataFrame**. This approach allows for easier data cleaning and preprocessing in subsequent steps.

```
# Create a DataFrame
data = pd.DataFrame({
    'Tweet': all_positive_tweets + all_negative_tweets,
    'Sentiment': [1] * len(all_positive_tweets) + [0] * len(all_negative_tweets)
})

data.head()
```

Here, I created a pandas DataFrame with two columns:

Tweet: This column contains all the tweets, both positive and negative.

Sentiment: This column holds the corresponding sentiment labels, where "1" denotes positive sentiment and "0" denotes negative sentiment.

By combining both sets of tweets into a single DataFrame, I facilitated the application of text-cleaning methods, as well as other processing steps that would follow. Additionally, organizing the data in a DataFrame makes it more straightforward to analyze and manipulate for training machine learning models later.

Finally, I printed the first few rows of the DataFrame to ensure that the data was correctly organized and ready for the next stages of the analysis.

Data Cleaning

Here how it looks the data before cleaning:

	Tweet	Sentiment
0	#FollowFriday @France_Inte @PKuchly57 @Milipol...	1
1	@Lamb2ja Hey James! How odd :/ Please call our...	1
2	@DespiteOfficial we had a listen last night :)...	1
3	@97sides CONGRATS :)	1
4	yeaaaah yippppy!!! my accnt verified rqst has...	1

To prepare the data for analysis, the next important step was to clean the tweets. Raw tweets often contain unnecessary elements like punctuation, emojis, and stopwords, which can negatively impact the performance of the machine learning models. In this section, I applied several preprocessing steps to clean the tweet text before using it for sentiment analysis.

Removing Emojis

First, I removed emojis using the emoji library with the function remove_emoji():

```
def remove_emoji(text):
    return emoji.replace_emoji(text, replace="")
```

Tokenization and Removing Punctuation: Next, I removed punctuation using str.maketrans() and tokenized the text with word_tokenize() from NLTK.

```
def cleaning(tweet):
    tweet = remove_emoji(tweet)
    tweet = tweet.translate(str.maketrans('', '', string.punctuation))
    words = word_tokenize(tweet)
```

Lowercasing and Removing Stopwords: I converted all words to lowercase and removed common stopwords like "the" and "in" using NLTK's stopwords list.

```
    words = [word.lower() for word in words if word.isalpha() and word not in stop_words]
```

Stemming: To reduce words to their root forms, I used the PorterStemmer to stem each word. I experimented with WordNetLemmatizer and LancasterStemmer if the results of model will get better, but it had no impact, or maybe with LancasterStemmer it had lower accuracy score. So, I decided to stay with PorterStemmer.

Joining Words: Finally, I joined the processed words back into a single string.

```
    stemmed_words = [stemmer.stem(word) for word in words]
    #stemmed_words = [lemmatizer.lemmatize(word) for word in words]

    return " ".join(stemmed_words)
```

After applying the cleaning function to each tweet, I added a new column to the dataframe called Cleared_Tweet:

```
data['Cleared_Tweet'] = data['Tweet'].apply(cleaning)
```

And now, that is how the DataFrame will look with the new column:

	Tweet	Sentiment	Cleared_Tweet
0	#FollowFriday @France_Inte @PKuchly57 @Milipol...	1	followfriday franceint milipolpari top engag m...
1	@Lamb2ja Hey James! How odd :/ Please call our...	1	hey jame how odd pleas call contact centr abl ...
2	@DespiteOfficial we had a listen last night :)...	1	despiteoffici listen last night as you bleed a...
3	@97sides CONGRATS :)	1	congrat
4	yeaaaah yippppy!!! my accnt verified rqst has...	1	yeaaaah yippppi accnt verifi rqst succeed got ...
5	@BhaktisBanter @PallaviRuhail This one is irre...	1	bhaktisbant pallaviruhail thi one irresist fli...
6	We don't like to keep our lovely customers wai...	1	we dont like keep love custom wait long we hop...
7	@Impatientraider On second thought, there's ju...	1	impatientraid on second thought enough time dd...
8	Jgh , but we have to go to Bayan :D bye	1	jgh go bayan d bye
9	As an act of mischievousness, am calling the E...	1	as act mischiev call etl layer inhous wareh ap...

Feature Extraction

Next, I separated the data into training and testing sets:

```
# Seperating data into training and testing sets
x = data['Cleared_Tweet']
y = data['Sentiment']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # shuffle=True is default
```

I used 80% of the data for training and 20% for testing, ensuring that the data was shuffled randomly (the default behavior of train_test_split).

To prepare the text data for machine learning models, I applied the TfidfVectorizer from sklearn, which converts the textual data into numerical format based on the Term Frequency-Inverse Document Frequency (TF-IDF) method. This transformation is important for model evaluation:

```
# Importing TfidfVectorizer from sklearn to convert text data into numerical format

from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

This step converts the cleaned text data (Cleared_Tweet) into a format suitable for model training, allowing the algorithms to better interpret the input data.

Model Selection, Training, and Evaluation

Naïve Bayes

To build the sentiment analysis model, I used the Multinomial Naive Bayes algorithm:

```
from sklearn.naive_bayes import MultinomialNB

# Final model after experimentation with alpha
model=MultinomialNB(alpha=10)
model.fit(X_train_vec, y_train)
```

I experimented with different values for the hyperparameter alpha during the model training process. The results showed that increasing or decreasing alpha did not improve the model's performance compared to using alpha=10.

After training, I evaluated the model's performance using accuracy score, classification report, and confusion matrix:

```
# Importing accuracy score, classification report and confusion matrix to evaluate the model perfomance

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

y_pred=model.predict(X_test_vec)

print("Accuracy Score:\n", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Model's Performance Results

Accuracy Score: The model achieved an accuracy of **75.15%**, meaning it correctly classified three-quarters of the tweets.

Confusion Matrix:

True Negatives (correct negative predictions): **788**.

False Positives (positive misclassified as negative): **200**.

True Positives (correct positive predictions): **715**.

False Negatives (negative misclassified as positive): **297**.

This shows the model handled negative tweets slightly better than positive ones.

Logistic Regression

The **Logistic Regression** model was implemented after experimenting with the parameters max_iter (set to **5000**) and regularization strength C (set to **1**).

Accuracy Score: The model achieved an accuracy of **76%**, slightly improving over the Naive Bayes model

```
from sklearn.linear_model import LogisticRegression

# Final model after experimentation with max_iter and C
model_2=LogisticRegression(C=1, max_iter=5000)
model_2.fit(X_train_vec,y_train)

y_pred_lr=model_2.predict(X_test_vec)

print("Accuracy Score:\n", accuracy_score(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test,y_pred_lr))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_lr))
```

Support Vector Machine (SVM) Model

The **SVM model** was implemented with the **RBF kernel**, after experimenting with parameters such as C (set to **1**) and gamma (set to **1**).

Accuracy Score: The model achieved an accuracy of **75.95%**, slightly below the performance of Logistic Regression.

```
from sklearn.svm import SVC

# Final model after experimentation with kernel, gamma and C
model_3 = SVC(C=1, kernel='rbf', gamma=1)
model_3.fit(X_train_vec,y_train)

y_pred_svc=model_3.predict(X_test_vec)

print("Accuracy Score:\n", accuracy_score(y_test, y_pred_svc))
print("Classification Report:\n", classification_report(y_test,y_pred_svc))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svc))
```

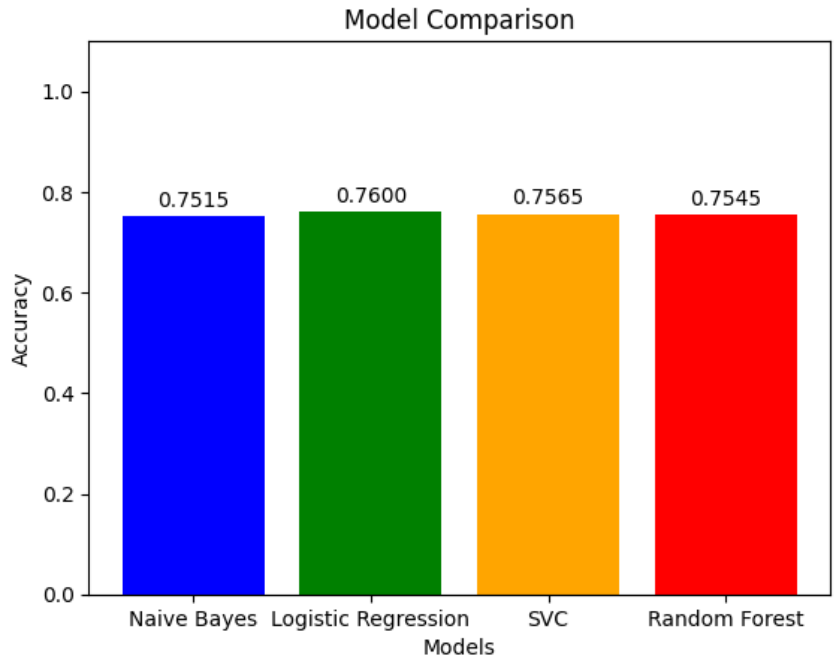
Random Forest Classifier

Using the RandomForestClassifier with n_estimators=1000, max_depth=100, and random_state=42, I experimented with various parameter values but couldn't achieve better performance than this configuration. The model achieved an accuracy score of **75.545** , and its metrics were evaluated using a classification report and confusion matrix.

```
from sklearn.ensemble import RandomForestClassifier

# Final model after experimentation with n_estimators and max_depth
model_4=RandomForestClassifier(n_estimators=1000, max_depth=100, random_state=42)
model_4.fit(X_train_vec,y_train)

y_pred_rf=model_4.predict(X_test_vec)

print("Accuracy Score:\n", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test,y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

Model Comparison Visualization

To better understand the model performances, I visualized the accuracy scores using a bar plot. The models compared were Naive Bayes, Logistic Regression, Support Vector Classifier (SVC), and Random Forest. Accuracy scores were displayed above each bar for clarity.



Conclusion

In conclusion, this project demonstrated how machine learning models can be applied to sentiment analysis, specifically using tweets. The dataset was preprocessed, features were extracted, and multiple models were tested. After evaluating the models**, Logistic Regression** emerged as the most effective, with an accuracy of **76%**. While **76%** might not seem very high in general, it's a good result for sentiment analysis, where it's harder to predict the meaning of the text. Also, a very high accuracy might suggest overfitting, where the model works well on the training data but not on new, unseen data.

This, being my first experience with Sentiment Analysis, I put in my best effort to achieve the most accurate prediction possible. It has sparked my interest to learn more and explore further projects in sentiment analysis and machine learning, expanding my knowledge and skills in the process.