INTERN PROJECT PHASE – 1

Project – 2

Twitter Sentiment Analysis

-JUJJAVARAPU SUJAN CHOWDARY

INTRODUCTION

Twitter Sentiment Analysis is an innovative data analytics project designed to delve into the vast realm of social media discourse. The project centers around the meticulous examination of a comprehensive dataset of tweets, with the primary goal of discerning the sentiments conveyed within each tweet whether they lean towards positivity, negativity, or neutrality. By leveraging sophisticated data analytics techniques, the project aspires to extract valuable insights into prevailing public opinions, emerging trends, and the dynamic spectrum of sentiments disseminated across the Twitter platform.

Sentiment analysis, a pivotal facet of this project, entails the identification and classification of sentiments expressed in textual sources. The unique nature of tweets, characterized by brevity and immediacy, positions them as an invaluable source for generating extensive sentiment data upon analysis. This treasure trove of data plays a crucial role in comprehending the collective opinions of individuals on a myriad of topics, making sentiment analysis a powerful tool for understanding public sentiment in the digital age.

The motivation behind this endeavor lies in the need to develop an Automated Machine Learning Sentiment Analysis Model. Twitter, as a microblogging platform, is replete with diverse content, but its efficacy for extracting meaningful insights is hindered by the presence of non-useful characters collectively termed as "noise." This noise, interspersed with valuable data, poses a significant challenge when implementing machine learning models. Therefore, the project seeks to surmount these challenges by creating a robust automated model capable of effectively computing customer perceptions through sentiment analysis, thereby contributing to a more nuanced understanding of the multifaceted world of Twitter discourse.

DATASET OVERVIEW

Context:

The dataset employed in this project is the sentiment140 dataset, a rich collection of 1,600,000 tweets extracted using the Twitter API. These tweets have been meticulously annotated, assigning sentiments as 0 for negative and 4 for positive, enabling the dataset's utility in sentiment detection. This vast corpus of Twitter data serves as the foundation for our Twitter Sentiment Analysis project.

Content:

The dataset comprises six key fields, each providing valuable insights into the characteristics of the tweets:

1. ***Target:***

   - Represents the polarity of the tweet.

   - Values: 0 for negative, 2 for neutral, and 4 for positive sentiments.

2. ***IDs:***

   - Denotes the unique identifier of each tweet.

   - Example: 2087

3. ***Date:***

   - Indicates the date and time when the tweet was posted.

   - Example: Sat May 16 23:58:44 UTC 2009

4. ***Flag:***

   - Represents the query associated with the tweet (e.g., lyx).

   - If there is no query, the value is marked as NO_QUERY.

5. ***User:***

   - Specifies the Twitter handle of the user who posted the tweet.

   - Example: robotickilldozr

6. ***Text:***

   - Encompasses the actual text content of the tweet.

   - Example: "Lyx is cool"

The diverse nature of this dataset, covering various sentiments and associated metadata, enables the development of a robust sentiment analysis model. The incorporation of target labels, user information, and timestamp data enhances the depth of analysis, providing a comprehensive understanding of sentiment dynamics within the Twitter platform. This dataset serves as the cornerstone for training and evaluating the Automated Machine Learning Sentiment Analysis Model, playing a pivotal role in extracting meaningful insights from the expansive world of Twitter discourse.

LINK: https://www.kaggle.com/datasets/kazanova/sentiment140

PROJECT OBJECTIVES

Data Exploration:

Exploratory Data Analysis:

In this crucial phase, our aim is to comprehensively understand the dataset, ensuring a solid foundation for subsequent analysis. We embark on a journey of data exploration, unraveling insights about the structure, content, and nuances of the imported data.

 # Display of the first 5 lines of our dataset:

df.head()

The initial exploration involves a glimpse of the first five records in the dataset, showcasing the target, IDs, date, flag, user, and text fields.

 # Display the column names of our dataset:

df.columns

The dataset comprises six columns: 'target', 'ids', 'date', 'flag', 'user', and 'text'.

 # Display the number of records in our dataset:

print('Length of data is', len(df))

The dataset consists of a substantial 1,600,000 records.

df.shape

The dataset is characterized by a shape of (1600000, 6), signifying 1,600,000 records and 6 columns.

df.info()

An in-depth look at the dataset's information reveals its structure, data types, and memory usage.

df.dtypes

The data types of the columns in our dataset are showcased, highlighting the presence of object types that necessitate further processing.

 # Checking for Null values:

np.sum(df.isnull().any(axis=1))

No missing values are detected in our dataset.

 # Rows and columns in the dataset:

print('Count of columns in the data is: ', len(df.columns))

print('Count of rows in the data is: ', len(df))

The dataset boasts 6 columns and 1,600,000 rows.

 # Checking unique Target Values:

df['target'].unique()

The 'target' column, crucial for sentiment analysis, contains only 0 and 4.

df['target'].nunique()

A count of unique values in the 'target' column reveals two distinct sentiments: 0 for negative sentiment and 4 for positive sentiment.

Data Visualization of Target Variables:

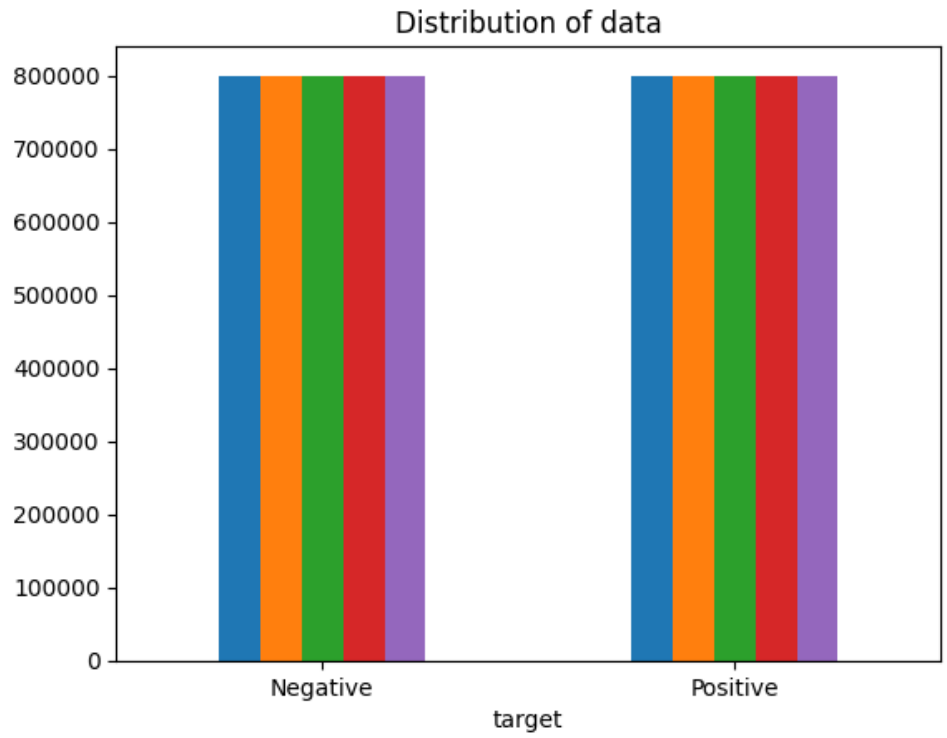
After meticulous data processing and focusing on the relevant columns, the exploration advances to visualizing target variables, providing a more intuitive understanding.

df.groupby('target').count()

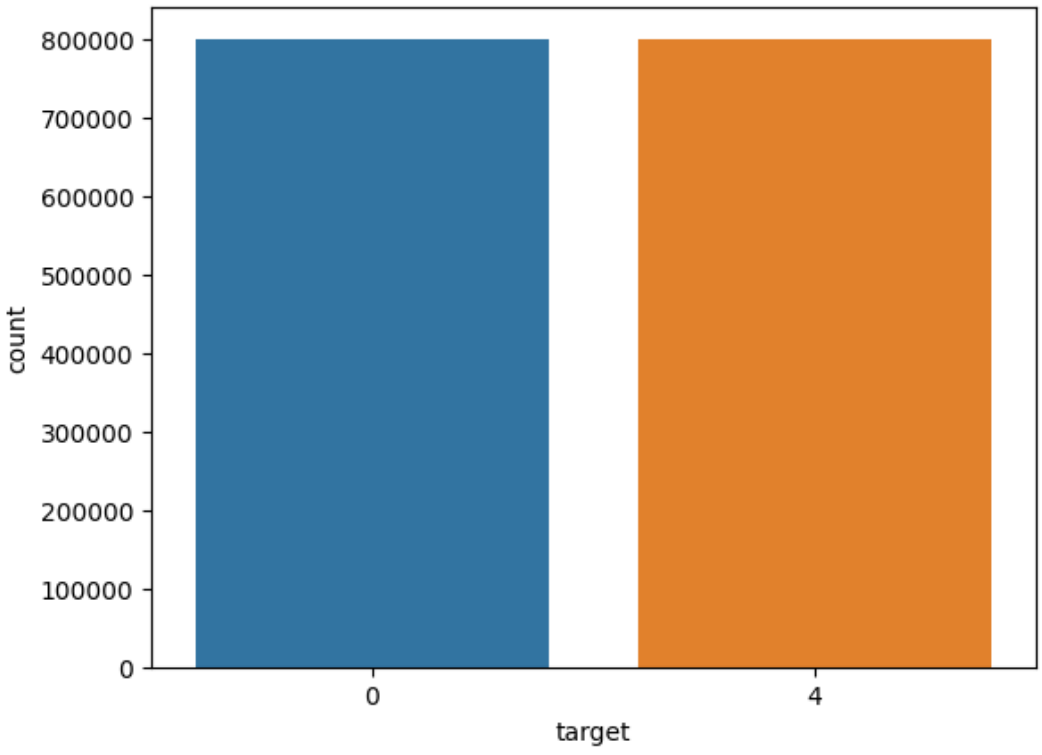
target	ids	date	flag	user	text
0	800000	800000	800000	800000	800000
4	800000	800000	800000	800000	800000

A groupby operation on the 'target' column showcases the distribution of sentiments, with equal counts for 0 (negative sentiment) and 4 (positive sentiment).

This thorough data exploration lays the groundwork for subsequent stages in our Twitter Sentiment Analysis project, facilitating informed decision-making and model development



- Each color represents one of the columns: **ids**, **date**, **flag**, **user** and **text**.
- **text** variable contains the **text** column.
- **sentiment** variable contains the **target** column.



.

- We did the same as before, we just used the**. countplot()** function from **seaborn**.

DATA PREPROCESSING

Before diving into the model training phase, extensive data preprocessing is imperative to ensure the quality and relevance of the data for subsequent analysis. The following steps are undertaken to refine the dataset:

1. Text Cleaning:

- Removal of stop words: Eliminating common words that may not contribute significantly to sentiment analysis.
- Removal of emojis: Extracting meaningful sentiments by excluding emojis.
- Conversion to lowercase: Standardizing text to lowercase for better generalization.
- Punctuation cleaning: Removing unnecessary noise by cleaning punctuation.
- Removal of repeating characters: Enhancing text clarity by eliminating repeated characters.
- Removal of URLs: Excluding URLs as they don't provide significant sentiment information.
- Removal of numeric numbers: Eliminating numerical digits from the text.

```
# Sample code for text cleaning
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)

def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)

def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))', ' ', data)

def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
```

```
# Applying the above cleaning functions to the 'text' column
dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords(text))
dataset['text'] = dataset['text'].apply(lambda x: cleaning_punctuations(x))
dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
```

2. Text Tokenization:

- Breaking down text into individual tokens (words).

```
# Sample code for text tokenization
dataset['text'] = dataset['text'].apply(word_tokenize)
```

3. Stemming and Lemmatization:

- Stemming: Reducing words to their derived stems for simplification.
- Lemmatization: Reducing derived words to their root form for improved results.

```
# Sample code for stemming and lemmatization
st = nltk.PorterStemmer()
lm = nltk.WordNetLemmatizer()

def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data

def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data
dataset['text'] = dataset['text'].apply(lambda x: stemming_on_text(x))
dataset['text'] = dataset['text'].apply(lambda x: lemmatizer_on_text(x))
```

4. Target Column Adjustment:

- Replacing target values to ease understanding.
- Mapping 4 to 1 for positive sentiment.

 # Sample code for target column adjustment

data['target'] = data['target'].replace(4,1)

5. Dataset Organization:

- Separating positive and negative tweets.
- Combining positive and negative tweets into a single dataset.

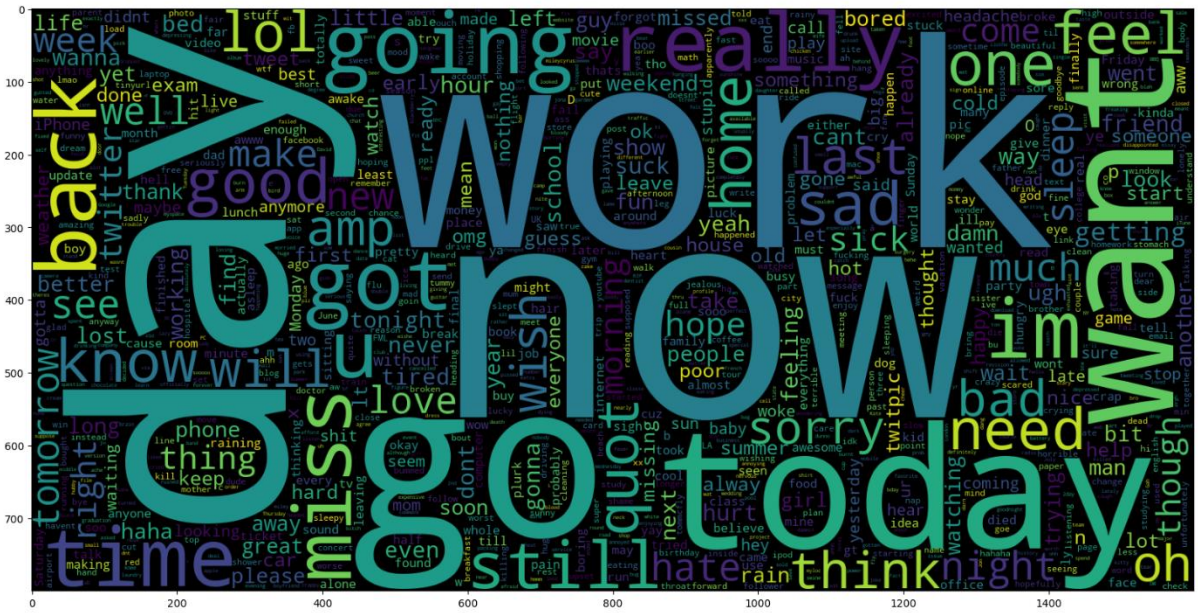
 # Sample code for dataset organization

data_pos = data[data['target'] == 1]

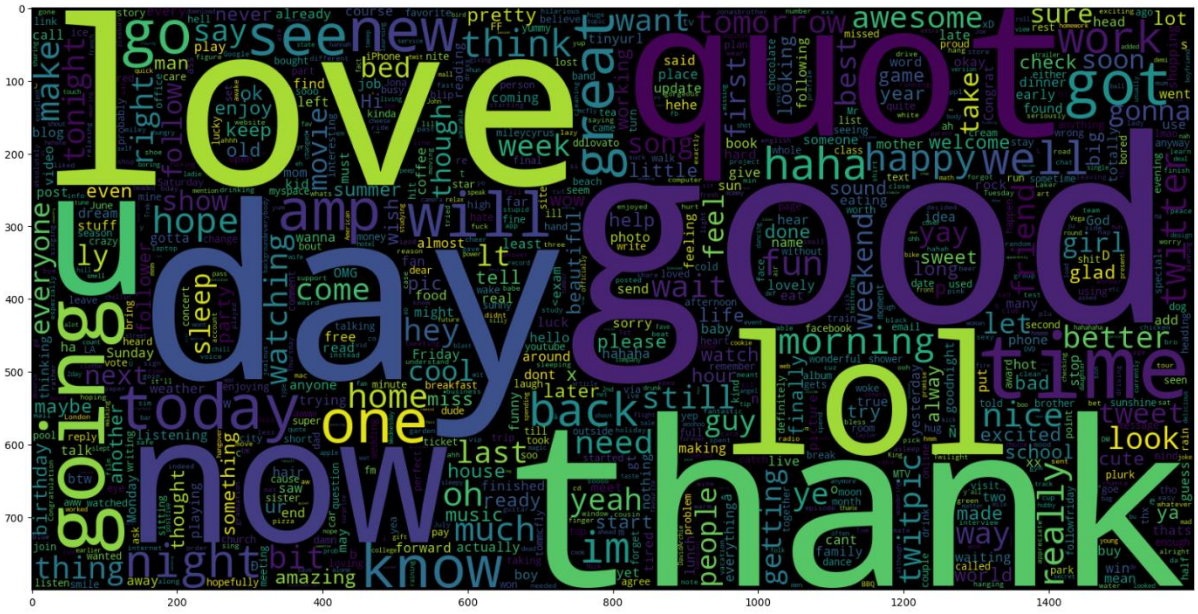
data_neg = data[data['target'] == 0]

dataset = pd.concat([data_pos, data_neg])

These comprehensive preprocessing steps equip the dataset for subsequent machine learning tasks, ensuring that the text data is refined and ready for analysis. The resulting dataset, now cleaned and tokenized, serves as a solid foundation for training a sentiment analysis model.



As the picture shows, a lot of negative words appear: bad, sad, wrong, etc



- As the picture shows, a lot of negative words appear: good, love, happy, etc

SPLITTING OUR DATA INTO TRAIN AND TEST SUBSET

In the model development process, it is crucial to split the dataset into training and testing subsets to assess the model's performance on unseen data. This step aids in evaluating the model's ability to generalize to new observations. Here's how we split the data and prepare it for further analysis:

 # Separating the 95% data for training data and 5% for testing data

X_train, X_test, y_train, y_test = train_test_split(dataset['text'], dataset['target'], test_size=0.05, random_state=26105111)

In this code snippet:

- `X` contains the 'text' data.

- `y` contains the 'target' data.

- `X_train` holds 95% of the 'text' data for training.

- `X_test` holds 5% of the 'text' data for testing.

- `y_train` contains 95% of the 'target' data for training.

- `y_test` contains 5% of the 'target' data for testing.

TRANSFORMING DATASET USING TF-IDF VECTORIZER

Text data needs to be transformed into a format suitable for machine learning models. The TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is utilized for this purpose.

 # Fit the TF-IDF Vectorizer:

vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)

vectoriser.fit(X_train)

```
TfidfVectorizer
TfidfVectorizer(max_features=500000, ngram_range=(1, 2))
```

 # Transform the data using TF-IDF Vectorizer:

X_train = vectoriser.transform(X_train)

X_test = vectoriser.transform(X_test)

In this code, the TF-IDF Vectorizer is trained on the training data (`X_train`) and applied to transform both the training and testing data (`X_train` and `X_test`).

FUNCTION FOR MODEL EVALUATION

After training the model, the next step is to evaluate its performance. A function, `model_Evaluate`, is defined for this purpose. It provides evaluation metrics such as accuracy, precision, recall, and the ROC-AUC curve. Additionally, it generates a confusion matrix with a heatmap for a visual representation of the model's performance.

	Positive	Negative
Positive	TP	TN
Negative	FP	TN

def model_Evaluate(model):

 # Predict values for Test dataset

y_pred = model.predict(X_test)

 # Print the evaluation metrics for the dataset.

print(classification_report(y_test, y_pred))

```
    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)
    categories = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten() /
np.sum(cf_matrix)]
    labels = [f'{v1}n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cf_matrix, annot=labels, cmap='Blues', fmt='',
                xticklabels=categories, yticklabels=categories)


    plt.xlabel("Predicted values", fontdict={'size':14}, labelpad=10)
    plt.ylabel("Actual values" , fontdict={'size':14}, labelpad=10)
    plt.title("Confusion Matrix", fontdict={'size':18}, pad=20)
```

This function provides a convenient way to evaluate models consistently by presenting key metrics and a visual representation of the model's classification performance.

MODEL BUILDING:

In addressing the sentiment analysis problem, three distinct models have been applied to the dataset to explore various classifier performances. The selected models range from simpler to more complex algorithms. Below are the details for each model:

Model-1: Bernoulli Naive Bayes (BNB)

```
BNBmodel = BernoulliNB()
start = time.time()
BNBmodel.fit(X_train, y_train)
end = time.time()
print("The execution time of this model is {:.2f} seconds\n".format(end-start))
model_Evaluate(BNBmodel)
y_pred1 = BNBmodel.predict(X_test)
```

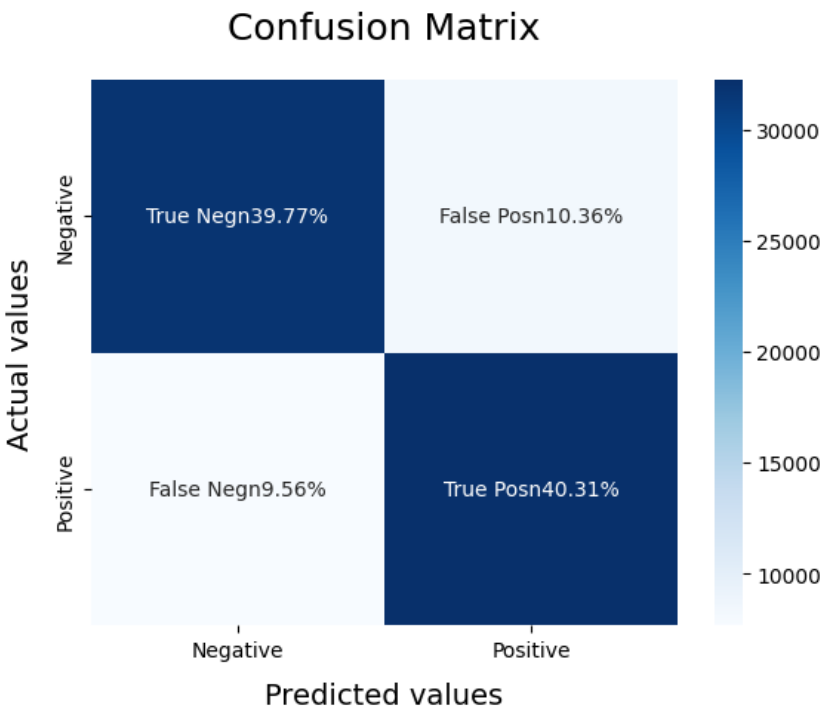
Model-1, Bernoulli Naive Bayes, is a probabilistic model that assumes features are binary-valued. The execution time for this model is 1.01 seconds.

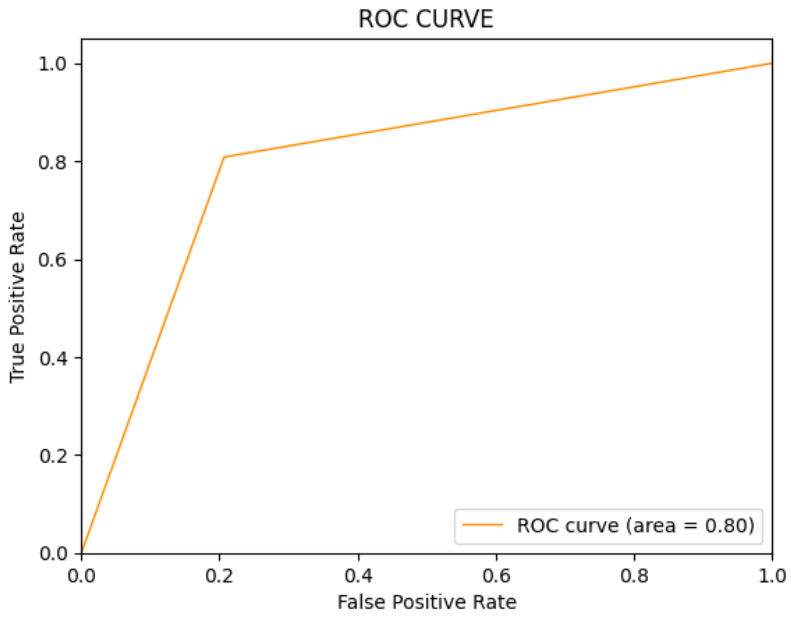
```
The execution time of this model is 1.01 seconds

              precision    recall  f1-score   support

           0       0.81      0.79      0.80     40100
           1       0.80      0.81      0.80     39900

    accuracy                           0.80     80000
   macro avg       0.80      0.80      0.80     80000
weighted avg       0.80      0.80      0.80     80000
```





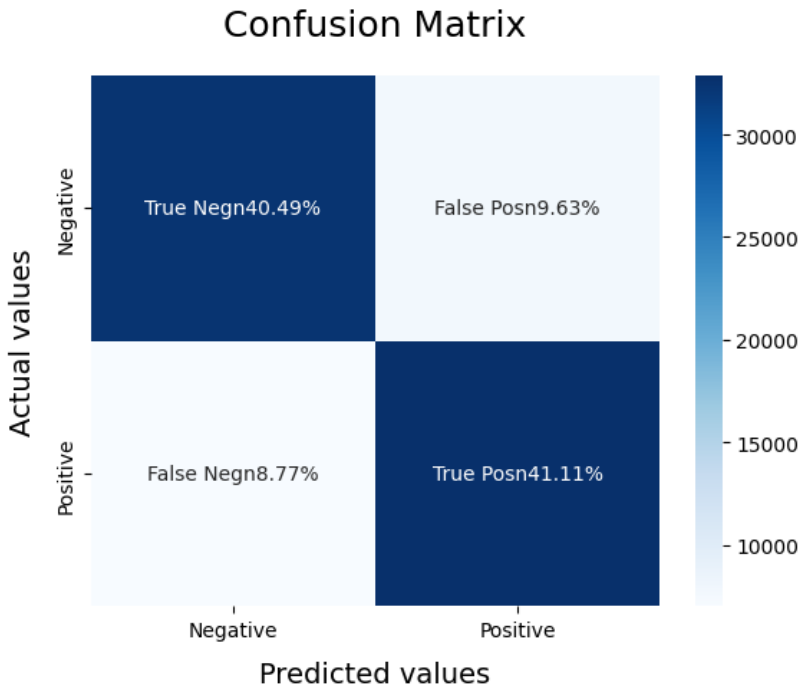
Model-2: Support Vector Machine (SVM)

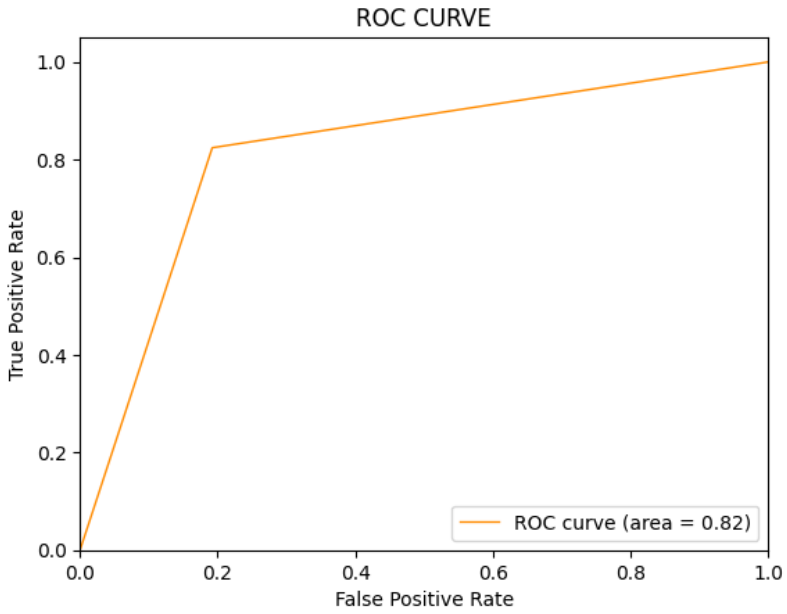
```
SVCmodel = LinearSVC()
start = time.time()
SVCmodel.fit(X_train, y_train)
end = time.time()
print("The execution time of this model is {:.2f} seconds\n".format(end-start))
model_Evaluate(SVCmodel)
y_pred2 = SVCmodel.predict(X_test)
```

Model-2 employs a Support Vector Machine (SVM), a powerful algorithm for classification tasks. The execution time for this model is 30.67 seconds.

```
The execution time of this model is 30.67 seconds
```

	precision	recall	f1-score	support
0	0.82	0.81	0.81	40100
1	0.81	0.82	0.82	39900
accuracy			0.82	80000
macro avg	0.82	0.82	0.82	80000
weighted avg	0.82	0.82	0.82	80000





Model-3: Logistic Regression

LRmodel = LogisticRegression(C=2, max_iter=1000, n_jobs=-1)

start = time.time()

LRmodel.fit(X_train, y_train)

end = time.time()

print("The execution time of this model is {:.2f} seconds\n".format(end-start))

model_Evaluate(LRmodel)

y_pred3 = LRmodel.predict(X_test)

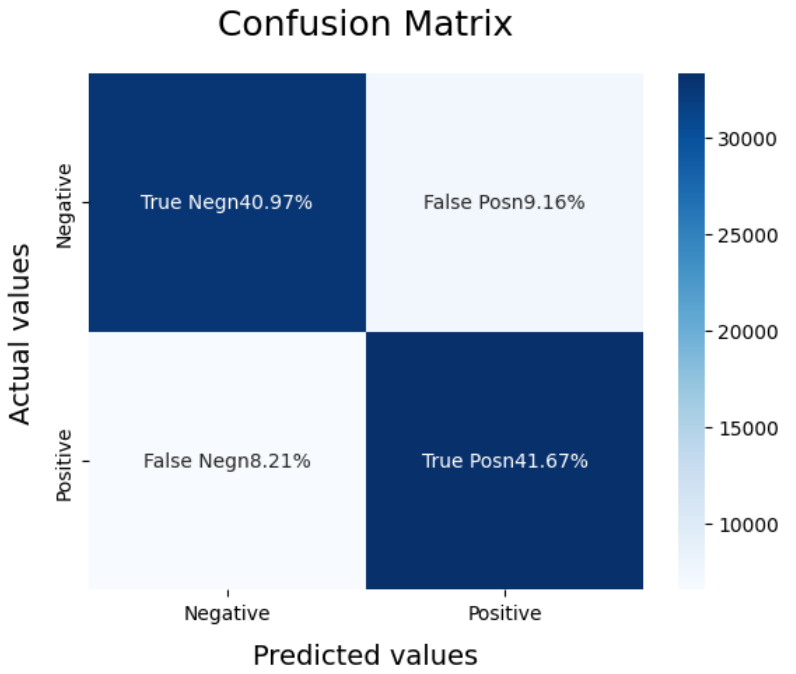
Model-3 utilizes Logistic Regression, a linear model suitable for binary classification. The execution time for this model is 138.79 seconds.

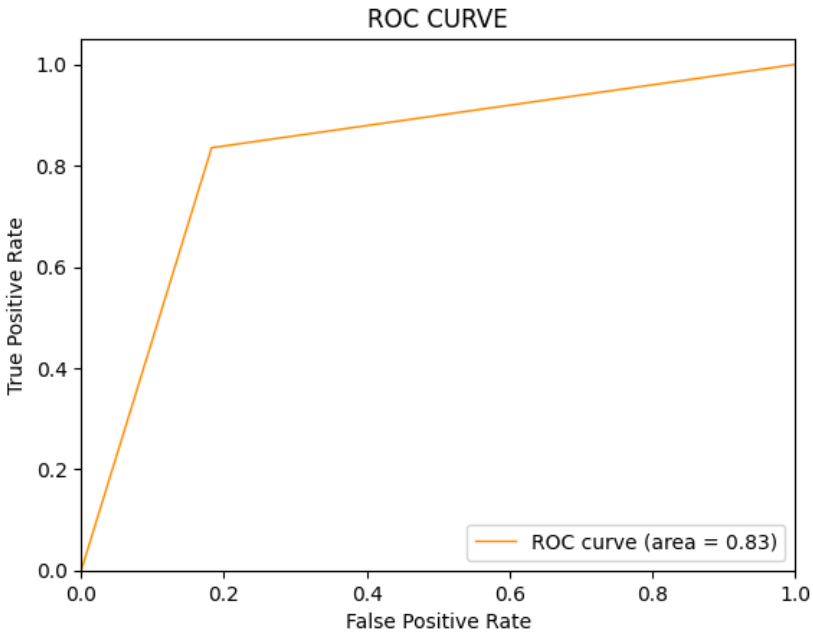
```
The execution time of this model is 138.79 seconds

              precision    recall  f1-score   support

           0       0.83      0.82      0.83     40100
           1       0.82      0.84      0.83     39900

    accuracy                           0.83     80000
   macro avg       0.83      0.83      0.83     80000
weighted avg       0.83      0.83      0.83     80000
```





Model Evaluation Metrics:

After training each model, we evaluate their performance using common metrics such as precision, recall, F1-score, and ROC-AUC curves.

- **Precision:** Indicates the accuracy of positive predictions.
- **Recall:** Measures the ability of the model to capture all positive instances.
- **F1-score:** The harmonic mean of precision and recall.
- **ROC-AUC Curve:** A graphical representation of the trade-off between true positive rate and false positive rate.

Visualizing ROC-AUC Curve for Each Model:

```
# Plot the ROC-AUC Curve for model-1, model-2, and model-3:
from sklearn.metrics import roc_curve, auc
fpr1, tpr1, thresholds1 = roc_curve(y_test, y_pred1)
roc_auc1 = auc(fpr1, tpr1)

fpr2, tpr2, thresholds2 = roc_curve(y_test, y_pred2)
roc_auc2 = auc(fpr2, tpr2)

fpr3, tpr3, thresholds3 = roc_curve(y_test, y_pred3)
roc_auc3 = auc(fpr3, tpr3)

plt.figure()
plt.plot(fpr1, tpr1, color='darkorange', lw=1, label='ROC curve - BNB (area = %0.2f)' % roc_auc1)
plt.plot(fpr2, tpr2, color='green', lw=1, label='ROC curve - SVM (area = %0.2f)' % roc_auc2)
plt.plot(fpr3, tpr3, color='blue', lw=1, label='ROC curve - LR (area = %0.2f)' % roc_auc3)

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE for Model Comparison')
plt.legend(loc="lower right")
plt.show()
```

The ROC-AUC curves provide a visual representation of each model's ability to distinguish between positive and negative sentiments. A larger area under the curve (AUC) signifies better performance.

RESULTS

After evaluating all models, we can conclude the following details

Model	Accuracy	F1-score (class 0)	F1-score (class 1)	AUC Score	Execution time
Bernoulli Naive Bayes (BNB)	80%	80%	80%	80%	0.69 seconds
Support Vector Machine (SVM)	82%	81%	82%	82%	28.32 seconds
Logistic Regression (LR)	83%	83%	83%	83%	163.56 seconds

- **Execution time**: When it comes to comparing the running time of models, Bernoulli Naive Bayes performs faster than SVM, which in turn runs faster than Logistic Regression.
- **Accuracy**: When it comes to model accuracy, logistic regression performs better than SVM, which in turn performs better than Bernoulli Naive Bayes.
- **F1-score**: The F1 Scores for **class 0** and **class 1** are:
    - For **class 0** (negative tweets):
        - accuracy: BNB (= 0.80) < SVM (=0.81) < LR (= 0.83)
    - For **class 1** (positive tweets):
        - accuracy: BNB (= 0.80) < SVM (=0.82) < LR (= 0.83)
- **AUC Score**: All three models have the same ROC-AUC score.
    - AUC score: BNB (= 0.80) < SVM (=0.82) < LR (= 0.83)
- We therefore conclude that **logistic regression** is the **best model** for the above dataset. (although it took much longer to run than other models).
- In our problem statement, **logistic regression** follows **Occam's razor principle** which defines that for a particular problem statement, if the data has no assumptions, then the simplest model works best. Since our **dataset has no assumptions** and **logistic regression is a simple model**, so the concept holds true for the dataset mentioned above.

FUTURE RECOMMENDATIONS:

- Further experimentation with advanced natural language processing techniques and deep learning models.
- Exploration of additional features or sentiment lexicons to enhance model performance.
- Continuous monitoring and updating of models to adapt to evolving language patterns on social media platforms.