CSCI544: Homework Assignment No. 1
Topic: Text Classification for Sentiment Analysis

Submitted by:
Yamini Haripriya Kanuparthi
kanupart@usc.edu

1. Dataset Preparation

The task begins by retrieving the dataset from the specified website using the 'requests' library. I used the 'pandas' package to read the ".gz" compressed file into a DataFrame without having to unzip it manually. Only the "review_body" and "star_rating" columns were required for sentiment analysis, so I preserved those.

To ensure data integrity and consistency, I removed any rows with missing values. After cleaning up the data, I renamed the columns 'Review' and 'Rating' for clarity. Finally, before analyzing, I examined the structure of my cleaned DataFrame and noted the number of NaN values in each column to confirm there was no missing data. Upon inspection, I detected and eliminated 97 NaN values from 'Reviews' and 14 from 'Ratings', yielding a clean dataset of 2,640,157 entries.

I created the binary labels for sentiment analysis. Ratings greater than three were considered positive sentiment and labeled as '1', whereas ratings of two or less were considered negative sentiment and labeled as '0'. This approach laid the groundwork for the later sentiment analysis.

Three sample reviews along with corresponding ratings

```
Three Sample Reviews and their Ratings:
                                          Review Rating
452767   My favorite memo books never fails me      5
940320                                    good      5
1012133                         good product      4
```

The statistics of the ratings, i.e., how many reviews received 1 ratings, etc.

```
Ratings Statistics:
Number of reviews with rating 1: 306968
Number of reviews with rating 2: 138381
Number of reviews with rating 3: 193686
Number of reviews with rating 4: 418360
Number of reviews with rating 5: 1582762
```

The number of reviews for each of these three classes (before discarding neutral reviews)

```
Number of reviews in each class before discarding the neutral reviews

Number of positive reviews (rating > 3): 2001122

Number of negative reviews (rating <= 2): 445349

Number of neutral reviews (rating == 3): 193686
```

Include the number of reviews for each of these three classes in your report (after discarding neutral)

```
Number of reviews in each class after discarding the neutral reviews

Number of positive reviews (rating > 3): 2001122

Number of negative reviews (rating <= 2): 445349

Number of neutral reviews (rating == 3): 0
```

2. Data Cleaning

During the data cleaning phase, I performed a number of preprocessing processes on the reviews dataset to prepare the text for analysis. I started by converting all reviews to lowercase to ensure uniformity and reduce complexity for the machine learning algorithms. Following that, I used BeautifulSoup to remove any HTML tags and regular expressions to remove URLs from the text, ensuring that only relevant textual material remained.

Continuing the cleaning process, I deleted any non-alphabetical characters to eliminate unnecessary punctuation and symbols that could distort the results. I also addressed the issue of excess spaces by utilizing regular expressions to detect and eliminate them, resulting in a more consistent text structure throughout all reviews. To address the complex element of contractions in the English language, I created a contraction expansion phase that converts shorter words like "won't" to their full forms, such as "will not," to improve text data quality for subsequent natural language processing activities with help of a contraction dictionary.

I calculated the average length of the reviews before and after the cleaning process, and discovered a little decrease from 319 to 303 characters on average. This quantifiable shift demonstrated successful pruning of unnecessary text, making the dataset more suited for sentiment analysis model training.

The average length of the reviews in terms of character length in the dataset before and after cleaning

```
Average review length: Before Data Cleaning: 319.01493 , After Data Cleaning: 303.88361
```

3. **Preprocessing**

During the preprocessing phase, I employed the NLTK package to further refine the dataset. I began by removing common stop words, which are typically abundant and bear little significance for sentiment analysis. This was followed by lemmatization, a process that reduces words to their base or dictionary form. I printed three sample reviews both before and after these preprocessing steps to illustrate the transformation and effectiveness of the text normalization. Finally, I calculated the average review length, which significantly decreased from 303 characters before preprocessing to 188 characters afterward, underscoring the impact of these cleaning procedures on the dataset.

Sample reviews before data cleaning + preprocessing.

```
Three sample reviews before data_cleaning + preprocessing:
1490405    My back requires me to work in a standing posi...
1512708    First of all, it was just thrown into a box th...
807100                                    grandson loves it.
Name: Review, dtype: object
```

Sample reviews after data cleaning + preprocessing.

```
Three sample reviews after data_cleaning + preprocessing:
125110    bought two first one tried either defective ba...
157477    much play door hit wall open fully issue insta...
170482    refill started skipping right away expected to...
Name: Review, dtype: object
```

The average length of the reviews in terms of character length in before and after preprocessing

```
Average review length - preprocessing: Before preprocessing: 303.88361 , After preprocessing: 188.722665
```

4. Feature Extraction

The TF-IDF vectorizer is used to extract features from the text. After applying the vectorizer to the input text, the resultant shape was (200000, 125394). The dataset was then divided into train and test groups (80:20) using the stratified sampling technique. By now, I've named my train and test sets (X_train, X_test, y_train, y_test).

For Questions 5-8, I used the "scikit-learn" library to train the models and then evaluated them on the train and test sets using classification evaluation metrics including precision, recall, and f1-score. All of the models use the default hyper-parameters.

5. Perceptron (Single Layered)

	Accuracy	Precision	Recall	F1 Score
Training Metrics	0.92035	0.9221509533432914	0.9182321546864649	0.9201873817903978
Testing Metrics	0.85415	0.8558431766775572	0.8516480768268894	0.8537404733253109

6. SVM (Linear)

	Accuracy	Precision	Recall	F1 Score
Training Metrics	0.93763125	0.9381679389312977	0.9370305098303898	0.9375988794186983
Testing Metrics	0.88975	0.8899064204573888	0.8894613114590106	0.8896838102861717

7. Logistic Regression

	Accuracy	Precision	Recall	F1 Score
Training Metrics	0.93150625	0.9323281907433381	0.9305685752496656	0.9314475519663712
Testing Metrics	0.892825	0.8952587074692974	0.8896613814835193	0.8924512681568451

8. Multinomial Naive Bayes

	Accuracy	Precision	Recall	F1 Score
Training Metrics	0.87778125	0.8968645781360783	0.8537627957553714	0.8747830875525872
Testing Metrics	0.8552	0.8735728941968748	0.8304906717351073	0.8514871794871794

In [1]:
```
import sys

print("Python version")
print(sys.version)
print("Version info.")
print(sys.version_info)
```

```
Python version
3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
Version info.
sys.version_info(major=3, minor=10, micro=12, releaselevel='final', serial=0)
```

In [2]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import requests
import io
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Read Data

In [5]:
```
path = "/content/drive/MyDrive/amazon_reviews_us_Office_Products_v1_00.tsv"
Sentiment_df = pd.read_csv(path, header=0, sep='\t', quotechar='"', on_bad_lines='skip')
```

```
<ipython-input-5-0c64d1b10dc6>:2: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or set low
_memory=False.
  Sentiment_df = pd.read_csv(path, header=0, sep='\t', quotechar='"', on_bad_lines='skip')
```

Keep Reviews and Ratings

In [6]:
```
columns_selected = ['review_body', 'star_rating']
Sentiment_data = Sentiment_df[columns_selected]
```

In [7]:
```
nan_check = Sentiment_data.isna().sum()
print("Number of NaN values in each column:")
print(nan_check)
```

```
Number of NaN values in each column:
review_body    97
star_rating    14
dtype: int64
```

In [8]:
```
Sentiment_data = Sentiment_data.dropna()
```

In [9]:
```
nan_check = Sentiment_data.isna().sum()
print(nan_check)
```

```
review_body    0
star_rating    0
dtype: int64
```

In [10]:
```
Sentiment_data.rename(columns={'review_body': 'Review', 'star_rating': 'Rating'}, inplace=True)
```

In [11]:
```
# Select three sample reviews
sample_reviews = Sentiment_data.sample(3)
print("Three Sample Reviews and their Ratings:")
print(sample_reviews)
```

```
Three Sample Reviews and their Ratings:
                                                   Review Rating
2567268  The phone was very annoying. Everytime I tried...      1
2561893  This phone is (for now) one of the most slende...      3
682869   I'll  never own another calendar without pocke...      5
```

In [12]:
```
Sentiment_data['Rating'] = pd.to_numeric(Sentiment_data['Rating'], errors='coerce')
```

In [13]:
```
nan_in_rating = Sentiment_data['Rating'].isna().sum()
print("Number of NaN values in 'Rating' column:", nan_in_rating)
```

```
Number of NaN values in 'Rating' column: 0
```

In [14]:
```
# Report statistics of the ratings
ratings_stats = Sentiment_data['Rating'].value_counts().sort_index()
print("\nRatings Statistics:")
for rating in range(1, 6):
    print(f"Number of reviews with rating {rating}: {ratings_stats.get(rating, 0)}")
```

```
Ratings Statistics:
Number of reviews with rating 1: 306968
Number of reviews with rating 2: 138381
Number of reviews with rating 3: 193686
Number of reviews with rating 4: 418360
Number of reviews with rating 5: 1582762
```

We form three classes and select 100000 reviews randomly from each class.

In [15]:
```
# Create labels for all three classes: 1 for positive (rating > 3), 0 for negative (rating <= 2), and -1 for neutral (ra
Sentiment_data['Sentiment'] = Sentiment_data['Rating'].apply(lambda x: 1 if x > 3 else (0 if x <= 2 else -1))
```

In [16]:
```
positive_count = (Sentiment_data['Sentiment'] == 1).sum()
negative_count = (Sentiment_data['Sentiment'] == 0).sum()
neutral_count = (Sentiment_data['Sentiment'] == -1).sum()
```

In [17]:
```
# Print the counts in seperate lines
print("Number of reviews in each class before discarding the neutral reviews")
print("\nNumber of positive reviews (rating > 3):", positive_count)
print("\nNumber of negative reviews (rating <= 2):", negative_count)
print("\nNumber of neutral reviews (rating == 3):", neutral_count)
```

```
Number of reviews in each class before discarding the neutral reviews

Number of positive reviews (rating > 3): 2001122

Number of negative reviews (rating <= 2): 445349

Number of neutral reviews (rating == 3): 193686
```

In [18]:
```
Sentiment_data.shape
```

Out[18]:
```
(2640157, 3)
```

In [19]:
```
Sentiment_data = Sentiment_data[Sentiment_data['Sentiment'] != -1]
```

In [20]:
```
neutral_count_after_removal = (Sentiment_data['Sentiment'] == -1).sum()
```

In [21]:
```
# Print the counts in seperate lines
print("\nNumber of reviews in each class after discarding the neutral reviews")
print("\nNumber of positive reviews (rating > 3):", positive_count)
```

```
print("\nNumber of negative reviews (rating <= 2):", negative_count)
print("\nNumber of neutral reviews (rating == 3):", neutral_count_after_removal)
```

```
Number of reviews in each class after discarding the neutral reviews

Number of positive reviews (rating > 3): 2001122

Number of negative reviews (rating <= 2): 445349

Number of neutral reviews (rating == 3): 0
```

In [22]:
```
Sentiment_data.shape
```

Out[22]:
```
(2446471, 3)
```

In [23]:
```
positive_reviews = Sentiment_data[Sentiment_data['Sentiment'] == 1].sample(n=100000, random_state=42)
negative_reviews = Sentiment_data[Sentiment_data['Sentiment'] == 0].sample(n=100000, random_state=42)

downsized_data = pd.concat([positive_reviews, negative_reviews])
downsized_data.reset_index(drop=True, inplace=True)
print("Shape of the downsized dataset:", downsized_data.shape)
```

```
Shape of the downsized dataset: (200000, 3)
```

In [24]:
```
avg_length_before = downsized_data['Review'].apply(len).mean()
```

Data Cleaning

In [25]:
```
downsized_data['Review'] = downsized_data['Review'].str.lower()
```

In [26]:
```
downsized_data['Review'] = downsized_data['Review'].apply(lambda x: BeautifulSoup(x, "html.parser").get_text())
```

```
<ipython-input-26-07e71b85188b>:1: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You
may want to open this file and pass the filehandle into Beautiful Soup.
  downsized_data['Review'] = downsized_data['Review'].apply(lambda x: BeautifulSoup(x, "html.parser").get_text())
```

In [27]:
```
downsized_data['Review'] = downsized_data['Review'].apply(lambda x: re.sub(r'http\S+', '', x))
```

In [28]:
```
contraction_dict = {
    "ain't": "am not / are not / is not / has not / have not",
    "aren't": "are not",
    "can't": "cannot",
```

```
    "can't've": "cannot have",
    "'cause": "because",
    "could've": "could have",
    "couldn't": "could not",
    "couldn't've": "could not have",
    "didn't": "did not",
    "doesn't": "does not",
    "don't": "do not",
    "hadn't": "had not",
    "hadn't've": "had not have",
    "hasn't": "has not",
    "haven't": "have not",
    "he'd": "he would / he had",
    "he'd've": "he would have",
    "he'll": "he will",
    "he'll've": "he will have",
    "he's": "he is / he has",
    "how'd": "how did",
    "how'd'y": "how do you",
    "how'll": "how will",
    "how's": "how is / how has / how does",
    "I'd": "I would / I had",
    "I'd've": "I would have",
    "I'll": "I will",
    "I'll've": "I will have",
    "I'm": "I am",
    "I've": "I have",
    "isn't": "is not",
    "it'd": "it would / it had",
    "it'd've": "it would have",
    "it'll": "it will",
    "it'll've": "it will have",
    "it's": "it is / it has",
    "let's": "let us",
    "ma'am": "madam",
    "might've": "might have",
    "mightn't": "might not",
    "mightn't've": "might not have",
    "must've": "must have",
    "mustn't": "must not",
    "mustn't've": "must not have",
    "needn't": "need not",
    "needn't've": "need not have",
    "o'clock": "of the clock",
    "oughtn't": "ought not",
```

```
    "oughtn't've": "ought not have",
    "shan't": "shall not",
    "sha'n't": "shall not",
    "shan't've": "shall not have",
    "she'd": "she would / she had",
    "she'd've": "she would have",
    "she'll": "she will",
    "she'll've": "she will have",
    "she's": "she is / she has",
    "should've": "should have",
    "shouldn't": "should not",
    "shouldn't've": "should not have",
    "so've": "so have",
    "so's": "so is / so as",
    "that'd": "that would / that had",
    "that'd've": "that would have",
    "that's": "that is / that has",
    "there'd": "there would / there had",
    "there'd've": "there would have",
    "there's": "there is / there has",
    "they'd": "they would / they had",
    "they'd've": "they would have",
    "they'll": "they will",
    "they'll've": "they will have",
    "they're": "they are",
    "they've": "they have",
    "to've": "to have",
    "wasn't": "was not",
    "we'd": "we would / we had",
    "we'd've": "we would have",
    "we'll": "we will",
    "we'll've": "we will have",
    "we're": "we are",
    "we've": "we have",
    "weren't": "were not",
    "what'll": "what will",
    "what'll've": "what will have",
    "what're": "what are",
    "what's": "what is / what has",
    "what've": "what have",
    "when's": "when is / when has",
    "when've": "when have",
    "where'd": "where did",
    "where's": "where is / where has",
    "where've": "where have",
```

```
    "who'll": "who will",
    "who'll've": "who will have",
    "who's": "who is / who has",
    "who've": "who have",
    "why's": "why is / why has",
    "why've": "why have",
    "will've": "will have",
    "won't": "will not",
    "won't've": "will not have",
    "would've": "would have",
    "wouldn't": "would not",
    "wouldn't've": "would not have",
    "y'all": "you all",
    "y'all'd": "you all would",
    "y'all'd've": "you all would have",
    "y'all're": "you all are",
    "y'all've": "you all have",
    "you'd": "you would / you had",
    "you'd've": "you would have",
    "you'll": "you will",
    "you'll've": "you will have",
    "you're": "you are",
    "you've": "you have"
}
```

In [29]:
```
def expand_contractions(text, contraction_dict):
    for contraction, expansion in contraction_dict.items():
        text = text.replace(contraction, expansion)
    return text
```

In [30]:
```
downsized_data['Review'] = downsized_data['Review'].apply(lambda x: expand_contractions(x, contraction_dict))
```

In [31]:
```
downsized_data['Review'] = downsized_data['Review'].str.replace(r'[^a-z\s]', '', regex=True)
```

In [32]:
```
downsized_data['Review'] = downsized_data['Review'].str.replace(r'\s+', ' ', regex=True)
```

In [33]:
```
avg_length_after = downsized_data['Review'].apply(len).mean()

# Print average review lengths before and after cleaning, separated by a comma
print("Average review length:",
      "Before Data Cleaning:", avg_length_before, ",",
      "After Data Cleaning:", avg_length_after)
```

```
Average review length: Before Data Cleaning: 319.01493 , After Data Cleaning: 303.88361
```

Pre-processing

In [34]:
```
avg_length_before_preprocessing_after_data_cleaning = downsized_data['Review'].apply(len).mean()
```

remove the stop words

In [35]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

In [36]:
```
nltk.download('stopwords')
nltk.download('punkt')

stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
    tokens = word_tokenize(text)
    filtered_tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(filtered_tokens)

downsized_data['Review'] = downsized_data['Review'].apply(remove_stopwords)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

perform lemmatization

In [37]:
```
from nltk.stem import WordNetLemmatizer
```

In [38]:
```
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

Out[38]:
```
True
```

In [39]:
```
lemmatizer = WordNetLemmatizer()

def lemmatize_text(text):
    tokens = word_tokenize(text)
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(lemmatized_tokens)

downsized_data['Review'] = downsized_data['Review'].apply(lemmatize_text)
```

In [40]:
```
# Print three sample reviews before preprocessing
print("\nThree sample reviews before data_cleaning + preprocessing:")
print(Sentiment_data['Review'].sample(3))
```

```
Three sample reviews before data_cleaning + preprocessing:
1542037    I like to have a way to quickly jot something ...
966437     Good when finally assembled but not machined c...
2464328    I bought two of these from Office Depot becaus...
Name: Review, dtype: object
```

In [41]:
```
# Print three sample reviews after preprocessing
print("\nThree sample reviews after data_cleaning + preprocessing:")
print(downsized_data['Review'].sample(3))
```

```
Three sample reviews after data_cleaning + preprocessing:
174109                                          good color
125786    purchased printer christmas present wife anoth...
18194                                             perfect
Name: Review, dtype: object
```

In [42]:
```
avg_length_after = downsized_data['Review'].apply(len).mean()
```

In [43]:
```
# Print average review lengths before and after preprocessing, separated by a comma
print("Average review length - preprocessing:",
      "Before preprocessing: ", avg_length_before_preprocessing_after_data_cleaning, ",",
      "After preprocessing:", avg_length_after)
```

```
Average review length - preprocessing: Before preprocessing:  303.88361 , After preprocessing: 188.722665
```

TF-IDF Feature Extraction

In [44]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf_vectorizer = TfidfVectorizer()

tfidf_features = tfidf_vectorizer.fit_transform(downsized_data['Review'])

features = tfidf_features
labels = downsized_data['Sentiment']

print("Shape of the TF-IDF feature matrix:", features.shape)
```

```
Shape of the TF-IDF feature matrix: (200000, 126233)
```

In [45]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.20, random_state=42)

print("Shape of X_train (features):", X_train.shape)
print("Shape of X_test (features):", X_test.shape)
print("Shape of y_train (labels):", y_train.shape)
print("Shape of y_test (labels):", y_test.shape)
```

```
Shape of X_train (features): (160000, 126233)
Shape of X_test (features): (40000, 126233)
Shape of y_train (labels): (160000,)
Shape of y_test (labels): (40000,)
```

Perceptron

In [46]:
```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

SentiAnlys_perceptron = Perceptron()

SentiAnlys_perceptron.fit(X_train, y_train)

y_train_pred = SentiAnlys_perceptron.predict(X_train)

y_test_pred = SentiAnlys_perceptron.predict(X_test)
```

In [47]:
```
print("Training Metrics for Single layer perceptron with coma between them:")
print("Accuracy:", accuracy_score(y_train, y_train_pred), end=", ")
print("Precision:", precision_score(y_train, y_train_pred), end=", ")
print("Recall:", recall_score(y_train, y_train_pred), end=", ")
print("F1 Score:", f1_score(y_train, y_train_pred))
```

```
Training Metrics for Single layer perceptron with coma between them:
Accuracy: 0.92035, Precision: 0.9221509533432914, Recall: 0.9182321546864649, F1 Score: 0.9201873817903978
```

In [48]:
```
print("\nTraining Metrics for Single layer perceptron with coma between them:")
print("Accuracy:", accuracy_score(y_test, y_test_pred), end=", ")
print("Precision:", precision_score(y_test, y_test_pred), end=", ")
print("Recall:", recall_score(y_test, y_test_pred), end=", ")
print("F1 Score:", f1_score(y_test, y_test_pred))
```

```
Training Metrics for Single layer perceptron with coma between them:
Accuracy: 0.85415, Precision: 0.8558431766775572, Recall: 0.8516480768268894, F1 Score: 0.8537404733253109
```

In [49]:
```
print("Printing the Performance Metrics for Single Layer Perceptron in seperate lines")
print("\nTraining Metrics:")
print("\nAccuracy:", accuracy_score(y_train, y_train_pred))
print("\nPrecision:", precision_score(y_train, y_train_pred))
print("\nRecall:", recall_score(y_train, y_train_pred))
print("\nF1 Score:", f1_score(y_train, y_train_pred))

print("\nTesting Metrics:")
print("\nAccuracy:", accuracy_score(y_test, y_test_pred))
print("\nPrecision:", precision_score(y_test, y_test_pred))
print("\nRecall:", recall_score(y_test, y_test_pred))
print("\nF1 Score:", f1_score(y_test, y_test_pred))
```

```
Printing the Performance Metrics for Single Layer Perceptron in seperate lines

Training Metrics:

Accuracy: 0.92035

Precision: 0.9221509533432914

Recall: 0.9182321546864649

F1 Score: 0.9201873817903978

Testing Metrics:

Accuracy: 0.85415

Precision: 0.8558431766775572

Recall: 0.8516480768268894

F1 Score: 0.8537404733253109
```

SVM

In [50]:
```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

SentiAnlys_linear_svm_model = LinearSVC()

SentiAnlys_linear_svm_model.fit(X_train, y_train)

y_train_pred_linear = SentiAnlys_linear_svm_model.predict(X_train)

y_test_pred_linear = SentiAnlys_linear_svm_model.predict(X_test)
```

In [51]:
```
print("Linear SVM Training Metrics:", end="")
print(" Accuracy:", accuracy_score(y_train, y_train_pred_linear), end=",")
print(" Precision:", precision_score(y_train, y_train_pred_linear), end=",")
print(" Recall:", recall_score(y_train, y_train_pred_linear), end=",")
print(" F1 Score:", f1_score(y_train, y_train_pred_linear))
```

```
Linear SVM Training Metrics: Accuracy: 0.9376375, Precision: 0.9381687126928708, Recall: 0.9370430087367355, F1 Score:
0.9376055228304505
```

In [52]:
```
print("\nLinear SVM Testing Metrics:", end="")
print(" Accuracy:", accuracy_score(y_test, y_test_pred_linear), end=",")
print(" Precision:", precision_score(y_test, y_test_pred_linear), end=",")
print(" Recall:", recall_score(y_test, y_test_pred_linear), end=",")
print(" F1 Score:", f1_score(y_test, y_test_pred_linear))
```

```
Linear SVM Testing Metrics: Accuracy: 0.88975, Precision: 0.8899064204573888, Recall: 0.8894613114590106, F1 Score: 0.8
896838102861717
```

In [53]:
```
print("Printing the Performance Metrics for Linear SVM in seperate lines")
print("\nTraining Metrics:")
print("\nAccuracy:", accuracy_score(y_train, y_train_pred_linear))
print("\nPrecision:", precision_score(y_train, y_train_pred_linear))
print("\nRecall:", recall_score(y_train, y_train_pred_linear))
print("\nF1 Score:", f1_score(y_train, y_train_pred_linear))

print("\nTesting Metrics:")
print("\nAccuracy:", accuracy_score(y_test, y_test_pred_linear))
print("\nPrecision:", precision_score(y_test, y_test_pred_linear))
print("\nRecall:", recall_score(y_test, y_test_pred_linear))
print("\nF1 Score:", f1_score(y_test, y_test_pred_linear))
```

```
Printing the Performance Metrics for Linear SVM in seperate lines

Training Metrics:

Accuracy: 0.9376375

Precision: 0.9381687126928708

Recall: 0.9370430087367355

F1 Score: 0.9376055228304505

Testing Metrics:

Accuracy: 0.88975

Precision: 0.8899064204573888

Recall: 0.8894613114590106

F1 Score: 0.8896838102861717
```

Logistic Regression

In [54]:
```
from sklearn.preprocessing import MaxAbsScaler

scaler = MaxAbsScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

In [55]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

SentiAnlys_logistic_model = LogisticRegression(max_iter=1000)  # Increase max_iter as needed

SentiAnlys_logistic_model.fit(X_train_scaled, y_train)

y_train_pred_logistic = SentiAnlys_logistic_model.predict(X_train_scaled)

y_test_pred_logistic = SentiAnlys_logistic_model.predict(X_test_scaled)
```

In [56]:
```
print("Logistic Regression Training Metrics:", end="")
print(" Accuracy:", accuracy_score(y_train, y_train_pred_logistic), end=",")
print(" Precision:", precision_score(y_train, y_train_pred_logistic), end=",")
print(" Recall:", recall_score(y_train, y_train_pred_logistic), end=",")
print(" F1 Score:", f1_score(y_train, y_train_pred_logistic))
```

```
Logistic Regression Training Metrics: Accuracy: 0.93150625, Precision: 0.9323281907433381, Recall: 0.9305685752496656,
F1 Score: 0.9314475519663712
```

In [57]:
```
print("\nLogistic Regression Testing Metrics:", end="")
print(" Accuracy:", accuracy_score(y_test, y_test_pred_logistic), end=",")
print(" Precision:", precision_score(y_test, y_test_pred_logistic), end=",")
print(" Recall:", recall_score(y_test, y_test_pred_logistic), end=",")
print(" F1 Score:", f1_score(y_test, y_test_pred_logistic))
```

```
Logistic Regression Testing Metrics: Accuracy: 0.892825, Precision: 0.8952587074692974, Recall: 0.8896613814835193, F1
Score: 0.8924512681568451
```

In [58]:
```
print("Printing the Performance Metrics for logistric regression in seperate lines")
print("\nTraining Metrics:")
print("\nAccuracy:", accuracy_score(y_train, y_train_pred_logistic))
```

```
print("\nPrecision:", precision_score(y_train, y_train_pred_logistic))
print("\nRecall:", recall_score(y_train, y_train_pred_logistic))
print("\nF1 Score:", f1_score(y_train, y_train_pred_logistic))

print("\nTesting Metrics:")
print("\nAccuracy:", accuracy_score(y_test, y_test_pred_logistic))
print("\nPrecision:", precision_score(y_test, y_test_pred_logistic))
print("\nRecall:", recall_score(y_test, y_test_pred_logistic))
print("\nF1 Score:", f1_score(y_test, y_test_pred_logistic))
```

```
Printing the Performance Metrics for logistric regression in seperate lines

Training Metrics:

Accuracy: 0.93150625

Precision: 0.9323281907433381

Recall: 0.9305685752496656

F1 Score: 0.9314475519663712

Testing Metrics:

Accuracy: 0.892825

Precision: 0.8952587074692974

Recall: 0.8896613814835193

F1 Score: 0.8924512681568451
```

Naive Bayes

In [59]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

SentiAnlys_nb_model = MultinomialNB()

SentiAnlys_nb_model.fit(X_train, y_train)

y_train_pred_nb = SentiAnlys_nb_model.predict(X_train)
```

```
y_test_pred_nb = SentiAnlys_nb_model.predict(X_test)
```

In [60]:
```
print("Multinomial Naive Bayes Training Metrics:", end="")
print(" Accuracy:", accuracy_score(y_train, y_train_pred_nb), end=",")
print(" Precision:", precision_score(y_train, y_train_pred_nb), end=",")
print(" Recall:", recall_score(y_train, y_train_pred_nb), end=",")
print(" F1 Score:", f1_score(y_train, y_train_pred_nb))
```

```
Multinomial Naive Bayes Training Metrics: Accuracy: 0.87778125, Precision: 0.8968645781360783, Recall: 0.85376279575537
14, F1 Score: 0.8747830875525872
```

In [61]:
```
print("\nMultinomial Naive Bayes Testing Metrics:", end="")
print(" Accuracy:", accuracy_score(y_test, y_test_pred_nb), end=",")
print(" Precision:", precision_score(y_test, y_test_pred_nb), end=",")
print(" Recall:", recall_score(y_test, y_test_pred_nb), end=",")
print(" F1 Score:", f1_score(y_test, y_test_pred_nb))
```

```
Multinomial Naive Bayes Testing Metrics: Accuracy: 0.8552, Precision: 0.8735728941968748, Recall: 0.8304906717351073, F
1 Score: 0.8514871794871794
```

In [62]:
```
print("Printing the Performance Metrics for Multinomial Naive Bayes in seperate lines")
print("\nTraining Metrics:")
print("\nAccuracy:", accuracy_score(y_train, y_train_pred_nb))
print("\nPrecision:", precision_score(y_train, y_train_pred_nb))
print("\nRecall:", recall_score(y_train, y_train_pred_nb))
print("\nF1 Score:", f1_score(y_train, y_train_pred_nb))

print("Testing Metrics:")
print("\nAccuracy:", accuracy_score(y_test, y_test_pred_nb))
print("\nPrecision:", precision_score(y_test, y_test_pred_nb))
print("\nRecall:", recall_score(y_test, y_test_pred_nb))
print("\nF1 Score:", f1_score(y_test, y_test_pred_nb))
```

```
Printing the Performance Metrics for Multinomial Naive Bayes in seperate lines

Training Metrics:

Accuracy: 0.87778125

Precision: 0.8968645781360783

Recall: 0.8537627957553714

F1 Score: 0.8747830875525872
Testing Metrics:

Accuracy: 0.8552

Precision: 0.8735728941968748

Recall: 0.8304906717351073

F1 Score: 0.8514871794871794
```

In [62]: