CSCI 544
Report: Assignment 1

1. Data Acquisition and Initial Handling

The dataset was retrieved from the provided link in the assignment guidelines. Once obtained, the data was unzipped, extracted, and stored locally. I employed the `read_table()` function from pandas to interpret the tsv file. Due to inconsistencies in certain rows, I had to exclude them during the reading phase. From the broad range of columns in the resultant file, the prime focus was on 'Reviews' (referred to as 'review_body') and 'Ratings' (denoted as 'star_ratings'). These columns were isolated and saved in the 'data' variable. Any further processing common to both the training and testing datasets will be executed before the division of the data.

2. Preliminary Data Cleansing

The data, being in its nascent form, required refinement. Prior to applying textual data-specific cleansing operations, a broader cleaning was executed. The attribute "star_ratings" consisted of diverse data types; hence, it was uniformly set to integer.

```
/var/folders/81/42j967ks3_75k4j8rqy841k40000gn/T/ipykernel_16877/3507883290.py:2: DtypeWarning: Columns (7) have mixe
d types. Specify dtype option on import or set low_memory=False.
  df = pd.read_table('amazon_reviews_us_Office_Products_v1_00.tsv',on_bad_lines='skip')
```

The dataset was assigned a new column 'label', which had the label (1 for rating <=3, 2 for rating>=4), as directed in the assignment. 50,000 records from each rating category were segregated. Using pandas' *concat()* method, these segments were merged to form a cohesive dataset of 100,000 rows, primed for textual cleaning. Following is a look at the dataset after categorizing:

```
training_dataset.head()
```

	star_rating	review_body	label
0	5	the phone case is awesome I've had other phon...	2
1	5	perfect	2
2	1	fast delivery... grand daughter likes it	1
3	5	This is a great product.	2
4	5	Works great and so much cheaper than buying at...	2

Textual Cleaning involved:

- Transforming text to lowercase using the *.lower()* method.
- Eliminating HTML elements and URLs with a custom function leveraging *BeautifulSoup*.
- Filtering out non-letter characters using the *re* library.

- Applying text contractions via user-defined function utilizing a long dictionary of keys (shortened English) to values (expanded English).

All of the above was set up in a function *clean_text()*, which was executed on every entry of the column 'review_body' of the dataframe using pandas' *apply()* function.

Measures of text length before and after cleaning:

- Average text length pre-cleaning: **319.24117**
- Average text length post-cleaning: **302.19097**

3. Stemming and Lemmatization:

Subsequent to cleaning, the textual data was further processed using tools from the nltk library:
- Tokenized the data (*nltk.tokenize.word_tokenize*).
- Removed stopwords (*nltk.corpus.stopwords*).
- Applied POS tagging (*nltk.pos_tag*).
- Performed Lemmatization on tagged data (*nltk.stem.WordNetLemmatizer*).

The same was executed on the data in a similar fashion to that of text cleaning.

Measures of text length before and after processing:

- Mean Text Length pre-processing: **302.19097**
- Mean Text Length post-processing: **185.10189**

Following is a look at the dataset after all textual processing:

```
training_dataset.head()
```

	star_rating	review_body	label
0	5	phone case awesome ive phone case didnt fit go...	2
1	5	perfect	2
2	1	fast delivery grand daughter like	1
3	5	great product	2
4	5	work great much cheap buy local store	2

4. Data Preparation for Modeling

Following stemming and lemmatization, 2 types of features were extracted from the dataset:
- Tf-IDF features (*sklearn.feature_extraction.text.TfidfVectorizer*)
- Bag Of Words (BOW) features (*sklearn.feature_extraction.text.CountVectorizer*)

Thus creating 2 training datasets. Allocating 80% to training and 20% to testing, the training sets, now labeled, were bifurcated accordingly. Hence, there were 2 sets of training features (tfidf and bow features and labels), and 2 sets of testing features.

The modeling phase utilized the sklearn library. It's noteworthy that to identify the optimal hyperparameters for each model, GridSearchCV was employed, extending the computation time.

After model training, outcomes were gauged in terms of precision, recall, and F1 scores for each category. The consolidated metrics were then logged in the format specified in the homework guidelines. Multiple iterations of model training introduced slight variations in results.
The summarized precision metrics acquired during model training are as follows:

```
Perceptron with tfidf:

Precision: 0.7888758323540932
Recall: 0.811933078008466
F1 Score: 0.8002384027018972

Perceptron with bow:

Precision: 0.7683394075320064
Recall: 0.8286635758919573
F1 Score: 0.7973621684526985

SVM with tfidf:

Precision: 0.8308837938467568
Recall: 0.8546663979036485
F1 Score: 0.8426073131955484

SVM with BOW:

Precision: 0.8560438402360628
Recall: 0.8186857488409595
F1 Score: 0.8369481221987533

LR with tfidf:

Precision: 0.8372877990668123
Recall: 0.8500302358395485
F1 Score: 0.8436109027256815

LR with BOW:

Precision: 0.8521766363921077
Recall: 0.82271719411409
F1 Score: 0.8371878365212041

MNB with tfidf:

Precision: 0.788416844845217
Recall: 0.8547671840354767
F1 Score: 0.8202524300014508
```

```
MNB with BOW:

Precision: 0.7679538072897871
Recall: 0.8578915541221528
F1 Score: 0.8104351137770162
```

The pages after this one are the pages of the jupyter notebook.

In [4]:
```
import sys
print(sys.version)
```

```
3.9.12 (main, Apr  5 2022, 01:53:17)
[Clang 12.0.0 ]
```

In [1]:
```
! pip install bs4
! pip install nltk
! pip install scikit-learn
# Dataset: https://web.archive.org/web/20201127142707if_/https://s3.amazona
```

```
Requirement already satisfied: bs4 in /Users/shreyavinaynayak/miniconda3/
lib/python3.10/site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in /Users/shreyavinaynayak/
miniconda3/lib/python3.10/site-packages (from bs4) (4.12.2)
Requirement already satisfied: soupsieve>1.2 in /Users/shreyavinaynayak/m
iniconda3/lib/python3.10/site-packages (from beautifulsoup4->bs4) (2.4.1)
Requirement already satisfied: nltk in /Users/shreyavinaynayak/miniconda
3/lib/python3.10/site-packages (3.8.1)
Requirement already satisfied: tqdm in /Users/shreyavinaynayak/miniconda
3/lib/python3.10/site-packages (from nltk) (4.65.0)
Requirement already satisfied: click in /Users/shreyavinaynayak/miniconda
3/lib/python3.10/site-packages (from nltk) (8.1.7)
Requirement already satisfied: regex>=2021.8.3 in /Users/shreyavinaynaya
k/miniconda3/lib/python3.10/site-packages (from nltk) (2023.8.8)
Requirement already satisfied: joblib in /Users/shreyavinaynayak/minicond
a3/lib/python3.10/site-packages (from nltk) (1.3.2)
Requirement already satisfied: scikit-learn in /Users/shreyavinaynayak/mi
niconda3/lib/python3.10/site-packages (1.3.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/shreyavinay
nayak/miniconda3/lib/python3.10/site-packages (from scikit-learn) (3.2.0)
Requirement already satisfied: joblib>=1.1.1 in /Users/shreyavinaynayak/m
iniconda3/lib/python3.10/site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: scipy>=1.5.0 in /Users/shreyavinaynayak/mi
niconda3/lib/python3.10/site-packages (from scikit-learn) (1.11.2)
Requirement already satisfied: numpy>=1.17.3 in /Users/shreyavinaynayak/m
iniconda3/lib/python3.10/site-packages (from scikit-learn) (1.25.2)
```

In [2]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('omw-1.4')
import re
from bs4 import BeautifulSoup
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus.reader.wordnet import NOUN, VERB, ADJ, ADV
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.linear_model import Perceptron,LogisticRegression
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC,LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/shreyavinaynayak/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/shreyavinaynayak/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/shreyavinaynayak/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/shreyavinaynayak/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/shreyavinaynayak/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Read Data

In [3]:
```
#Reading the tsv file
df = pd.read_table('data.tsv',on_bad_lines='skip')
```

```
/var/folders/81/42j967ks3_75k4j8rqy841k40000gn/T/ipykernel_16877/35078832
90.py:2: DtypeWarning: Columns (7) have mixed types. Specify dtype option
on import or set low_memory=False.
  df = pd.read_table('amazon_reviews_us_Office_Products_v1_00.tsv',on_bad
_lines='skip')
```

In [4]:
```
df.head()
```

Out[4]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	pro
0	US	43081963	R18RVCKGH1SSI9	B001BM2MAC	307809868	Scotch Cushion Wrap 7961, 12 Inches x 100 Feet	
1	US	10951564	R3L4L6LW1PUOFY	B00DZYEXPQ	75004341	Dust-Off Compressed Gas Duster, Pack of 4	
2	US	21143145	R2J8AWXWTDX2TF	B00RTMUHDW	529689027	Amram Tagger Standard Tag Attaching Tagging Gu...	
3	US	52782374	R1PR37BR7G3M6A	B00D7H8XB6	868449945	AmazonBasics 12-Sheet High-Security Micro-Cut ...	
4	US	24045652	R3BDDDZMZBZDPU	B001XCWP34	33521401	Derwent Colored Pencils, Inktense Ink Pencils,...	

Keep Reviews and Ratings

In [5]:
```
data = df[['star_rating','review_body']] #keeping only columns needed
data.dropna(axis=0,inplace=True)
```

```
/var/folders/81/42j967ks3_75k4j8rqy841k40000gn/T/ipykernel_16877/39527863
98.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data.dropna(axis=0,inplace=True)
```

In [6]:
```
data.head()
```

Out[6]:

	star_rating	review_body
0	5	Great product.
1	5	What's to say about this commodity item except...
2	5	Haven't used yet, but I am sure I will like it.
3	1	Although this was labeled as "new" the...
4	4	Gorgeous colors and easy to use

In [7]:
```
data['star_rating'].value_counts()
```

Out[7]:
```
5    1458992
4     389603
1     286072
3     179867
2     129031
5     123770
4      28757
1      20896
3      13819
2       9350
Name: star_rating, dtype: int64
```

In [8]:
```
data['star_rating']=data['star_rating'].astype('int')
```

```
/var/folders/81/42j967ks3_75k4j8rqy841k40000gn/T/ipykernel_16877/39771728
07.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data['star_rating']=data['star_rating'].astype('int')
```

In [9]:

```
#splitting data into classes

class1 = data[data['star_rating']<=3] #defining class 1 for ratings with va
labels = [1]*len(class1)
class1['label'] = labels

class2 = data[data['star_rating']>=4]  #defining class 2 for ratings with v
labels = [2]*len(class2)
class2['label'] = labels
```

```
/var/folders/81/42j967ks3_75k4j8rqy841k40000gn/T/ipykernel_16877/27453493
82.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  class1['label'] = labels
/var/folders/81/42j967ks3_75k4j8rqy841k40000gn/T/ipykernel_16877/27453493
82.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  class2['label'] = labels
```

We form two classes and select 50000 reviews randomly from each class.

In [10]:

```
# Sampling 50,000 random reviews from each class
sampled_class1 = class1.sample(n=50000, random_state=42)  # Using a fixed r
sampled_class2 = class2.sample(n=50000, random_state=42)
# Concatenating the sampled data to create a balanced dataset
balanced_data = pd.concat([sampled_class1, sampled_class2], ignore_index=Tr
# Shuffle the dataset
training_dataset = balanced_data.sample(frac=1, random_state=42).reset_inde
```

In [11]:

```
del df
```

In [12]:
```
training_dataset.head()
```

Out[12]:

	star_rating	review_body	label
0	5	the phone case is awesome I've had other phon...	2
1	5	perfect	2
2	1	fast delivery... grand daughter likes it	1
3	5	This is a great product.	2
4	5	Works great and so much cheaper than buying at...	2

Data Cleaning

Pre-processing

In [13]:
```
def extract_text_from_html(text):
    # Removing <style> and <script> tags and their content
    text = re.sub(r'<style.*?>.*?</style>', '', text, flags=re.DOTALL)
    text = re.sub(r'<script.*?>.*?</script>', '', text, flags=re.DOTALL)
    # Removing <a> tags, keeping their inner text
    text = re.sub(r'<a.*?>(.*?)</a>', r'\1', text)
    # Removing any remaining HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Joining the stripped strings
    text = ' '.join(text.strip().split())
    return text
```

In [14]:
```
def contract_text(text):
    contractions_dict = {
        "ain't": "am 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'd've": "he would have",
        "he'll": "he will",
        "he'll've": "he will have",
        "he's": "he is",
        "how'd": "how did",
        "how'd'y": "how do you",
        "how'll": "how will",
        "how's": "how is",
        "I'd": "I would",
        "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'd've": "it would have",
        "it'll": "it will",
        "it'll've": "it will have",
        "it's": "it is",
        "let's": "let us",
        "ma'am": "madam",
        "mayn't": "may not",
        "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'd've": "she would have",
```

```
        "she'll": "she will",
        "she'll've": "she will have",
        "she's": "she is",
        "should've": "should have",
        "shouldn't": "should not",
        "shouldn't've": "should not have",
        "so've": "so have",
        "so's": "so is",
        "that'd": "that would",
        "that'd've": "that would have",
        "that's": "that is",
        "there'd": "there would",
        "there'd've": "there would have",
        "there's": "there is",
        "they'd": "they would",
        "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'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've": "what have",
        "when's": "when is",
        "when've": "when have",
        "where'd": "where did",
        "where's": "where is",
        "where've": "where have",
        "who'll": "who will",
        "who'll've": "who will have",
        "who's": "who is",
        "who've": "who have",
        "why's": "why is",
        "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'd've": "you would have",
        "you'll": "you will",
        "you'll've": "you will have",
        "you're": "you are",
        "you've": "you have"
    }

    for contraction, expansion in contractions_dict.items():
        text = text.replace(contraction, expansion)

    return text
```

In [15]:
```
# testing function
contract_text("I won't be sad")
```

Out[15]: 'I will not be sad'

In [16]:
```
def clean_text(text):
    # making text lowercase
    text = text.lower()
    # removing HTML
    text = extract_text_from_html(text)
    # removing non-english characters and unnecessary spaces
    text = re.sub(r'[^a-zA-Z\s]', '', text).strip()
    # performing contractions
    text = contract_text(text)
    return text

def process_row(row):
    cleaned_text = clean_text(row['review_body'])
    row['review_body'] = cleaned_text
    return row
# applying the above functions on the training dataset:
average_length_before = training_dataset['review_body'].apply(len).mean()
training_dataset = training_dataset.apply(process_row, axis=1, result_type=
average_length_after =training_dataset['review_body'].apply(len).mean()

print("Average text length before cleaning: ",average_length_before)
print("Average text length after cleaning: ",average_length_after)
```

```
Average text length before cleaning:  319.24117
Average text length after cleaning:  302.19097
```

remove the stop words

In [17]:
```
def filter_out_stopwords(input_text):
    english_stops = set(stopwords.words('english'))
    tokenized_words = word_tokenize(input_text)
    filtered_words = []
    for word in tokenized_words:
        if word.lower() not in english_stops:
            filtered_words.append(word)

    return ' '.join(filtered_words)
```

In [18]:
```
def get_wordnet_pos(treebank_tag):
    pos_mapping = {
        'J': ADJ,
        'V': VERB,
        'N': NOUN,
        'R': ADV
    }
    return pos_mapping.get(treebank_tag[0], NOUN)
```

perform lemmatization

In [19]:
```
def lemmatize(text):
    result = []
    text = nltk.pos_tag(word_tokenize(text))
    lem = WordNetLemmatizer()
    for word in text:
        result.append(lem.lemmatize(word[0],get_wordnet_pos(word[-1])))
    return ' '.join(result)
```

In [20]:
```
def remove_stop_word_and_lemmatize(text):
    #removing stopwords
    text = filter_out_stopwords(text)
    #performing lemmatization
    text = lemmatize(text)
    return text

def stop_word_lemmatization(row):
    cleaned_text = remove_stop_word_and_lemmatize(row['review_body'])
    row['review_body'] = cleaned_text
    return row
# applying the above function to the training dataset:
average_length_before = training_dataset['review_body'].apply(len).mean()
training_dataset = training_dataset.apply(stop_word_lemmatization, axis=1,
average_length_after =training_dataset['review_body'].apply(len).mean()


print("Average text length before cleaning: ",average_length_before)
print("Average text length after cleaning: ",average_length_after)
```

```
Average text length before cleaning:  302.19097
Average text length after cleaning:  185.10189
```

In [21]:
```
training_dataset.head()
```

Out[21]:

	star_rating	review_body	label
0	5	phone case awesome ive phone case didnt fit go...	2
1	5	perfect	2
2	1	fast delivery grand daughter like	1
3	5	great product	2
4	5	work great much cheap buy local store	2

TF-IDF and BoW Feature Extraction

In [22]:
```
# TF IDF
# initialize TfidfVectorizer
vectorizer = TfidfVectorizer()
frequency_matrix = vectorizer.fit_transform(training_dataset['review_body']
```

In [23]:
```
# tfidf = pd.DataFrame.sparse.from_spmatrix(frequency_matrix, columns=vecto
# tfidf['label'] = training_dataset['label']
```

In [24]:
```
# tfidf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 79351 entries, aa to zz
dtypes: Sparse[float64, 0](79350), int64(1)
memory usage: 27.6 MB
```

In [25]:
```
# Bag Of Words Implementation
# Initialize CountVectorizer
count_vectorizer = CountVectorizer()
# Fit and transform the reviews
bow_features = count_vectorizer.fit_transform(training_dataset['review_body

# # Create a DataFrame for BoW features
# bow_df = pd.DataFrame.sparse.from_spmatrix(bow_features, columns=count_ve
# bow_df['label'] = training_dataset['label']

# # print("Bag-of-Words Features")
# # print(bow_df)
# bow_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 79351 entries, aa to zz
dtypes: Sparse[int64, 0](79350), int64(1)
memory usage: 27.6 MB
```

In [26]:
```
#splitting datasets into training and testing sets:
X_train_tf, X_test_tf, y_train_tf, y_test_tf = train_test_split(frequency_m
X_train_bow, X_test_bow, y_train_bow, y_test_bow = train_test_split(bow_fea
```

Perceptron Using Both Features

In [29]:
```
param_grid = {
    'eta0': [2e-6,3e-6,6e-6,7e-6,7e-7,7e-10,8e-7,0.00001],
    'max_iter': [700,1000,2000],#5000,10000
    'penalty': [None, 'l2', 'l1', 'elasticnet'],
    'alpha': [0.012,0.0131,0.0132,0.0133,0.0134,0.135,0.014,0.015]
}

grid= GridSearchCV(Perceptron(), param_grid, scoring = 'f1')
grid.fit(X_train_bow,y_train_bow)
best_percep = grid.best_estimator_
percep_preds_bow = best_percep.predict(X_test_bow)

precision = precision_score(y_test_bow, percep_preds_bow)
recall = recall_score(y_test_bow, percep_preds_bow)
f1 = f1_score(y_test_bow, percep_preds_bow)
print("Perceptron with bow: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
Perceptron with bow:
Precision: 0.7683394075320064 Recall: 0.8286635758919573 F1 Score: 0.7973
621684526985
```

In [30]:
```
print(best_percep)
```

```
Perceptron(alpha=0.012, eta0=1e-05, max_iter=700, penalty='l2')
```

In [27]:
```
param_grid = {
    'eta0': [0.05,0.03,0.01, 0.001,0.003],
    'max_iter': [5000,10000], #10000,
    'l1_ratio':[0,0.02,0.05, 0.1,0.125,0.15,0.175,0.25],
    'penalty': ['elasticnet'], #'l2', 'l1',
    'alpha': [1e-6,5e-6,1e-5,5e-5]
}
grid= GridSearchCV(Perceptron(), param_grid, scoring = 'f1')
grid.fit(X_train_tf,y_train_tf)
best_percep = grid.best_estimator_
percep_preds_tf = best_percep.predict(X_test_tf)

precision = precision_score(y_test_tf, percep_preds_tf)
recall = recall_score(y_test_tf, percep_preds_tf)
f1 = f1_score(y_test_tf, percep_preds_tf)
print("Perceptron with tfidf: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
Perceptron with tfidf:
Precision: 0.7888758323540932 Recall: 0.811933078008466 F1 Score: 0.80023
84027018972
```

In [28]:
```
print(best_percep)
```

```
Perceptron(alpha=1e-05, eta0=0.01, max_iter=5000, penalty='elasticnet')
```

SVM Using Both Features

In [33]:
```
param_grid= {
    'C': [0.001, 0.005, 0.01, 0.012, 0.015, 0.017, 0.02, 0.025],
    'tol': [1e-6, 5e-6, 1e-5, 5e-5, 1e-4],
    'max_iter': [1000000],
    'intercept_scaling': [0.9, 0.95, 1.0, 1.05, 1.1]
}

svm = LinearSVC()

grid_search = GridSearchCV(svm, param_grid, scoring='f1', cv=5)  # Using 5-
grid_search.fit(X_train_bow, y_train_bow)
best_svm = grid_search.best_estimator_
svm_preds_bow = best_svm.predict(X_test_bow)

precision = precision_score(y_test_bow, svm_preds_bow)
recall = recall_score(y_test_bow, svm_preds_bow)
f1 = f1_score(y_test_bow, svm_preds_bow)
print("SVM with BOW: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
SVM with BOW:
Precision: 0.8560438402360628 Recall: 0.8186857488409595 F1 Score: 0.8369
481221987533
```

In [34]:
```
print(best_svm)
```

```
LinearSVC(C=0.02, intercept_scaling=0.9, max_iter=1000000, tol=1e-06)
```

In [31]:
```
param_grid = {
    'C': [0.001, 0.005, 0.015, 0.025],
    'max_iter': [1000000],
    'tol': [0.00001],
    'intercept_scaling': [1.0, 1.25, 1.5, 1.75, 2.0, 2.25]
}

svm = LinearSVC()
grid_search = GridSearchCV(svm, param_grid, scoring='f1', cv=5)  # Using 5-
grid_search.fit(X_train_tf, y_train_tf)
best_svm = grid_search.best_estimator_
svm_preds_tf = best_svm.predict(X_test_tf)

precision = precision_score(y_test_tf, svm_preds_tf)
recall = recall_score(y_test_tf, svm_preds_tf)
f1 = f1_score(y_test_tf, svm_preds_tf)
print("SVM with tfidf: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
SVM with tfidf:
Precision: 0.8308837938467568 Recall: 0.8546663979036485 F1 Score: 0.8426
073131955484
```

In [32]:
```
print(best_svm)
```

```
LinearSVC(C=0.025, intercept_scaling=1.0, max_iter=1000000, tol=1e-05)
```

Logistic Regression Using Both Features

In [37]:
```
param_grid = {
    'C': [0.1, 0.25, 0.5, 1.0],
    'solver': ['lbfgs','liblinear','saga'],
    'max_iter': [10000,100000]
}

grid_search = GridSearchCV(LogisticRegression(), param_grid, scoring='f1')
grid_search.fit(X_train_bow, y_train_bow)

best_lr = grid_search.best_estimator_
lr_preds_bow = best_lr.predict(X_test_bow)

precision = precision_score(y_test_bow, lr_preds_bow)
recall = recall_score(y_test_bow, lr_preds_bow)
f1 = f1_score(y_test_bow, lr_preds_bow)
print("LR with BOW: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
LR with BOW: 
Precision: 0.8521766363921077 Recall: 0.82271719411409 F1 Score: 0.837187
8365212041
```

In [38]:
```
print(best_lr)
```

```
LogisticRegression(C=0.25, max_iter=10000, solver='liblinear')
```

In [35]:
```
# Logistic regression
param_grid = {
    'C': [0.5, 0.75, 1.0, 1.5, 2.0, 3.0],
    'solver': ['lbfgs','liblinear','saga'],
    'max_iter': [10000,100000]
}

grid_search = GridSearchCV(LogisticRegression(), param_grid, scoring='f1')
grid_search.fit(X_train_tf, y_train_tf)

best_lr = grid_search.best_estimator_
lr_preds_tf = best_lr.predict(X_test_tf)

precision = precision_score(y_test_tf, lr_preds_tf)
recall = recall_score(y_test_tf, lr_preds_tf)
f1 = f1_score(y_test_tf, lr_preds_tf)
print("LR with tfidf: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
LR with tfidf: 
Precision: 0.8372877990668123 Recall: 0.8500302358395485 F1 Score: 0.8436
109027256815
```

In [36]:
```
print(best_lr)
```

```
LogisticRegression(C=1.5, max_iter=10000, solver='liblinear')
```

Naive Bayes Using Both Features

In [41]:
```
# Naive Bayes
param_grid = { 'alpha': [1.0,2.5,5.5,5.75,5.95,6.0,6.15,6.25, 6.5,10.0]}
grid_search = GridSearchCV(MultinomialNB(), param_grid, scoring='f1',cv=10)
grid_search.fit(X_train_bow, y_train_bow)

best_mnb = grid_search.best_estimator_

mnb_preds_bow = best_mnb.predict(X_test_bow)

precision = precision_score(y_test_bow, mnb_preds_bow)
recall = recall_score(y_test_bow, mnb_preds_bow)
f1 = f1_score(y_test_bow, mnb_preds_bow)
print("MNB with BOW: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
MNB with BOW: 
Precision: 0.7679538072897871 Recall: 0.8578915541221528 F1 Score: 0.8104
351137770162
```

In [42]:
```
print(best_mnb)
```

```
MultinomialNB(alpha=6.0)
```

In [39]:
```
# Naive Bayes
param_grid = { 'alpha': [0.001, 0.01, 0.1, 0.5, 1.0, 1.5, 2.0, 5.0, 10.0,10

grid_search = GridSearchCV(MultinomialNB(), param_grid, scoring='f1',cv=10)
grid_search.fit(X_train_tf, y_train_tf)

best_mnb = grid_search.best_estimator_

# Evaluate on the test set
mnb_preds_tf = best_mnb.predict(X_test_tf)

precision = precision_score(y_test_tf, mnb_preds_tf)
recall = recall_score(y_test_tf, mnb_preds_tf)
f1 = f1_score(y_test_tf, mnb_preds_tf)
print("MNB with tfidf: ")
print(f"Precision: {precision} Recall: {recall} F1 Score: {f1}")
```

```
MNB with tfidf:
Precision: 0.788416844845217 Recall: 0.8547671840354767 F1 Score: 0.82025
24300014508
```

In [40]:
```
print(best_mnb)
```

```
MultinomialNB()
```

In [ ]:

In [ ]: