CSCI544: Homework Assignment №1

ADITYA DUTTA

1. Dataset Preparation

#Loading Data:

```
import pandas as pd


import pandas as pd
from sklearn.model_selection import train_test_split

import pandas as pd

file_path =
r"C:\Users\adity\Desktop\NLP\amazon_reviews_us_Office_Products_v1_00.tsv"
data = pd.read_csv(file_path, sep='\t', on_bad_lines="skip")
print (data)

data = data[['review_body', 'star_rating']]

data = data[data['star_rating'] !=3]
```

#Removing rows with non-numeric 'star_rating' values:

```
data = data[pd.to_numeric(data['star_rating'], errors='coerce').notnull()]
```

#Converting 'star_rating' to integers:

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

#Function:

```
data['Sentiment'] = data['star_rating'].apply(lambda rating : +1 if rating
> 3 else 0)

print (data)
```

#Printing no. of reviews for each sentiment:

```
print('Number of positive reviews:', len(data[data['Sentiment'] == 1]))
print('Number of negative reviews:', len(data[data['Sentiment'] == 0]))
```

# Selecting 100,000 positive reviews and 100,000 negative reviews:

```
data_pos = data[data['Sentiment'] == 1].sample(n=100000)
data_neg = data[data['Sentiment'] == 0].sample(n=100000)
```

# Concatenating the reviews:

```
data2 = pd.concat([data_pos, data_neg])
print(data2)

train_data, test_data = train_test_split(data2, test_size=0.2,
random_state=42)
```

#Splitting into training and testing sets:

```
print('Number of reviews in the training set:', len(train_data))
print('Number of reviews in the testing set:', len(test_data))
```

2.Data Cleaning

```
import re
def clean_text(text):

    text = str(text)
```

# Convert text to lower case:

```
        text = text.lower()
```

# Remove HTML tags:

```
        text = re.sub(r'<.*?>', '', text)
```

# Remove URLs:

```
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
```

# Remove non-alphabetical characters:

```
         text = re.sub(r'[^a-zA-Z\s]', '', text)
```

# Remove extra spaces:

```
        text = re.sub(r'\s+', ' ', text).strip()

         return text
```

#Apply Cleaning:

```
data2['cleaned_review'] = data2['review_body'].apply(clean_text)
```

# Average review length before cleaning

```
data2['review_length_before'] = data2['review_body'].astype(str).apply(len)
avg_length_before = data2['review_length_before'].mean()
```

# Applying the cleaning function:

```
data2['cleaned_review'] = data2['review_body'].apply(clean_text)
```

# Average review length after cleaning:

```
data2['review_length_after'] = data2['cleaned_review'].apply(len)
avg_length_after = data2['review_length_after'].mean()

print("Average length before cleaning:", avg_length_before)
print("Average length after cleaning:", avg_length_after)
```

3.Pre-processing

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

nltk.download('stopwords')
nltk.download('wordnet')
```

#Defining Stop words and Lemmatizer:

```
lemmatizer = WordNetLemmatizer()

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

def preprocess_reviews(reviews):
```

#Tokenization:

```
reviews = reviews.str.split()
```

#Removing stop words and perform lemmatization :

```
 reviews = reviews.apply(lambda x: [lemmatizer.lemmatize(word) for word in
x if word not in stop_words])
```

#Joining back into single string:

```
reviews = reviews.str.join(' ')

    return reviews
```

# Print 3 sample reviews before preprocessing:

```
print('Before preprocessing:')
print(data2['review_body'].head(3))

data2['review_body'] = data2['review_body'].astype(str)
```

# Average length of the reviews before preprocessing:

```
average_length_before = data2['review_body'].apply(len).mean()
```

```
print('Average length before preprocessing:', average_length_before)

data2['review_body'] = preprocess_reviews(data2['review_body'])
```

# Print 3 sample reviews after preprocessing:

```
print('After preprocessing:')
print(data2['review_body'].head(3))
```

# Average length of the reviews after preprocessing:

```
average_length_after = data2['review_body'].apply(len).mean()
print('Average length after preprocessing:', average_length_after)
```

4. TF-IDF Feature Extraction

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

# Initialize the TF-IDF vectorizer:

```
vectorizer = TfidfVectorizer()
```

#Transform the reviews into TF-IDF features:

```
tfidf_features = vectorizer.fit_transform(data2['review_body'])
```

# Print the shape of the TF-IDF features:

```
print('Shape of TF-IDF features:', tfidf_features.shape)
```

5.Perceptron Model

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.model_selection import train_test_split
```

# Split the dataset into training and testing sets:

```
X_train, X_test, y_train, y_test = train_test_split(tfidf_features,
data2['Sentiment'], test_size=0.2, random_state=42)


perceptron = Perceptron()
```

# Training the model:

```
perceptron.fit(X_train, y_train)
```

# Using on training and testing data:

```
y_train_pred = perceptron.predict(X_train)
y_test_pred = perceptron.predict(X_test)


train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_precision = precision_score(y_train, y_train_pred)
test_precision = precision_score(y_test, y_test_pred)
train_recall = recall_score(y_train, y_train_pred)
test_recall = recall_score(y_test, y_test_pred)
train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)
```

# Print:

```
print("Perceptron Training Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
print("Recall:", train_recall)
print("F1 Score:", train_f1)

print("\nPerceptron Testing Metrics:")
print("Accuracy:", test_accuracy)
print("Precision:", test_precision)
print("Recall:", test_recall)
print("F1 Score:", test_f1)
```

6.Logistic Regression and Multinomial Naïve Bayes

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB


log_reg_model = LogisticRegression(max_iter=1000)
nb_model = MultinomialNB()


log_reg_model.fit(X_train, y_train)
nb_model.fit(X_train, y_train)


models = {'Logistic Regression': log_reg_model, 'Naive Bayes': nb_model}

for name, model in models.items():
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    print(f"{name} Training Metrics:")
    print("Accuracy:", accuracy_score(y_train, y_train_pred))
    print("Precision:", precision_score(y_train, y_train_pred))
    print("Recall:", recall_score(y_train, y_train_pred))
    print("F1 Score:", f1_score(y_train, y_train_pred))

    print(f"\n{name} Testing Metrics:")
    print("Accuracy:", accuracy_score(y_test, y_test_pred))
    print("Precision:", precision_score(y_test, y_test_pred))
    print("Recall:", recall_score(y_test, y_test_pred))
    print("F1 Score:", f1_score(y_test, y_test_pred))
    print("\n")
```

7. SVM Model

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

model = SVC()

model.fit(X_train, y_train)
```

# Labels for the training and testing data:

```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

# Metrics for the training data:

```
print('Training data:')
print('Accuracy:', accuracy_score(y_train, y_train_pred))
print('Precision:', precision_score(y_train, y_train_pred))
print('Recall:', recall_score(y_train, y_train_pred))
print('F1 score:', f1_score(y_train, y_train_pred))
```

# Metrics for the testing data

```
print('Testing data:')
print('Accuracy:', accuracy_score(y_test, y_test_pred))
print('Precision:', precision_score(y_test, y_test_pred))
print('Recall:', recall_score(y_test, y_test_pred))
print('F1 score:', f1_score(y_test, y_test_pred))
```