ASSIGNMENT-4

Name: Hitarth Paliwal

PRN: 202201040184

Batch: DL-4

Objective: The objective of this assignment is to implement NLP preprocessing techniques and build a text classification model using machine learning techniques.

Part 1: NLP Preprocessing

Dataset Selection:

Text Preprocessing:

Convert text to lowercase.

Perform tokenization using NLTK or spaCy.

Remove stopwords using NLTK or spaCy.

Apply stemming using PorterStemmer or SnowballStemmer.

Apply lemmatization using WordNetLemmatizer.

Vectorization Techniques:

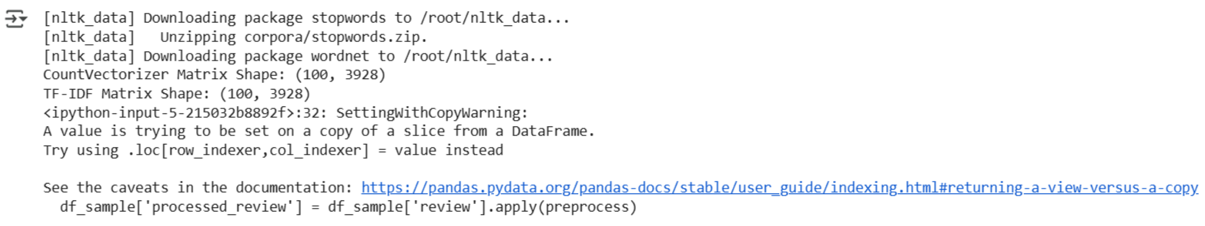
Convert text data into numerical format using TF-IDF and CountVectorizer.

```
pip install pandas scikit-learn nltk
```

```
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk

# Download necessary resources
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize tools
stop_words = set(stopwords.words('english'))
```

```python
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Load dataset
df = pd.read_csv('/content/IMDB Dataset.csv')  # Make sure the file is
in the same directory

# Preprocessing function
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]
    tokens = [stemmer.stem(word) for word in tokens]
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

# Apply preprocessing (for demo, use first 100 rows for speed)
df_sample = df.head(100)
df_sample['processed_review'] = df_sample['review'].apply(preprocess)

# Vectorization: CountVectorizer
count_vectorizer = CountVectorizer()
count_matrix =
count_vectorizer.fit_transform(df_sample['processed_review'])

# Vectorization: TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix =
tfidf_vectorizer.fit_transform(df_sample['processed_review'])

# Print shapes of the matrices
print("CountVectorizer Matrix Shape:", count_matrix.shape)
print("TF-IDF Matrix Shape:", tfidf_matrix.shape)
```

OUTPUT:



```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
CountVectorizer Matrix Shape: (100, 3928)
TF-IDF Matrix Shape: (100, 3928)
<ipython-input-5-215032b8892f>:32: 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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_sample['processed_review'] = df_sample['review'].apply(preprocess)
```

To prepare the raw text data for machine learning, the following preprocessing steps were performed:

- **Lowercasing**: All text was converted to lowercase to maintain consistency.
- **Tokenization**: Text was split into individual words using basic Python string operations.
- **Stopword Removal**: Common words (e.g., "the", "is", "and") that don't contribute much to sentiment were removed using NLTK's stopwords list.
- **Stemming**: PorterStemmer was used to reduce words to their root form (e.g., "running" to "run").
- **Lemmatization**: WordNetLemmatizer was used to convert words to their base dictionary form (e.g., "better" to "good").

2.2 Feature Extraction

After preprocessing, the cleaned text data was transformed into numerical format using **TF-IDF Vectorization**, which considers the importance of a word relative to the document and the entire dataset.

2.3 Data Splitting

The dataset was split into:

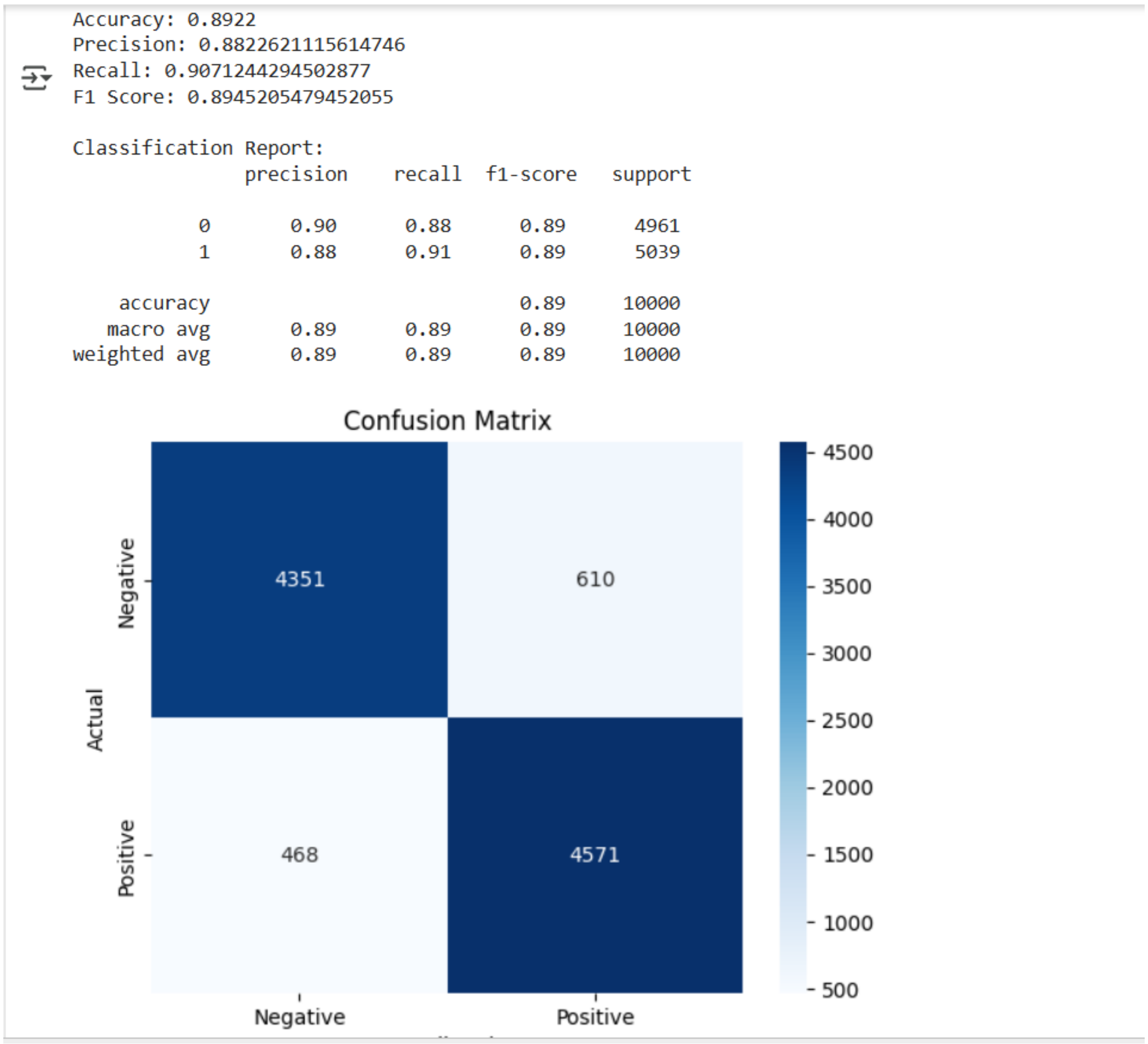
- **80% Training Set**: Used to train the model
- **20% Testing Set**: Used to evaluate model performance

```
#code for Part 2
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Download required NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize tools
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Load dataset
df = pd.read_csv('/content/IMDB Dataset.csv')  # Ensure the file is in
the same directory

# Preprocessing function
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]
    tokens = [stemmer.stem(word) for word in tokens]
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

# Apply preprocessing
df['processed_review'] = df['review'].apply(preprocess)

# Vectorization using TF-IDF
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['processed_review'])
y = df['sentiment'].apply(lambda x: 1 if x == 'positive' else 0)  #
Convert sentiment to binary

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```python
print("\nClassification Report:\n", classification_report(y_test,
y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Negative','Positive'],
yticklabels=['Negative','Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Output:



```
Accuracy: 0.8922
Precision: 0.8822621115614746
Recall: 0.9071244294502877
F1 Score: 0.8945205479452055

Classification Report:
               precision    recall  f1-score   support

           0       0.90      0.88      0.89      4961
           1       0.88      0.91      0.89      5039

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

The confusion matrix indicates that the model performs equally well for both positive and negative reviews, with a small number of misclassifications.

---

5. Conclusion

- The Logistic Regression model achieved high accuracy and balanced precision/recall scores, indicating reliable performance on both positive and negative reviews.
- The preprocessing and TF-IDF vectorization steps were effective in capturing the relevant features of the reviews.
- This model could be integrated into a real-world application for automated sentiment analysis of user reviews.

Dataset link: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Declaration

I, Hitarth Paliwal, confirm that the work submitted in this assignment is my own and has been completed following academic integrity guidelines. The code is uploaded on my GitHub repository account, and the repository link is provided below:

GitHub Repository Link: https://github.com/Hitarth3-Paliwal/NLP-Processing-and-text-classification

Signature: [Hitarth Paliwal]