Programming Assignment 3: Sentiment Analysis

1. Preprocessing

Several modules and resources in NLTK library is imported for natural language processing tasks. stop_words is created to be used for filtering out common words. WordNetLemmatizer() will be used to reduce words to their base form.

preprocess_text() function is defined to perform several preprocessing steps on the content in the training and testing data. It first converts the text to lowercase by .lower() and tokenizes the text into individual words by word_tokenize() in NLTK. Then it filters out non-alphabetic tokens and stop words, and lemmatizes each remaining token by WordNetLemmatizer() in NLTK. Finally, it joins the filtered tokens back into a single string, separated by spaces, and returns the preprocessed text.

```
[ ] def preprocess_text(text):
        tokens = word_tokenize(text.lower())
        filtered_tokens = [lemmatizer.lemmatize(token) for token in tokens if token.isalpha() and token not in stop_words]
        return ' '.join(filtered_tokens)
```

After the preprocess_text() function is defined, it can be applied to the contents in the training and testing data.

2. Find the Best Classifier

(1) Split the data into training and validation sets

To evaluate the models, we split the original training data into training and validation sets. 20% of the data will be used for validation.

(2) Feature extraction

After splitting the data, we convert the text data into a numerical representation using the Term Frequency-Inverse Document Frequency (TF-IDF) scheme. vectorizer.fit_transform(X_train) applies the vectorizer to the training data (X_train), transforming it into a matrix where each row represents a document and each column represents a unique word. The values in the matrix correspond to the TF-IDF scores of each word in each document. vectorizer.transform(X_valid) applies the same vectorizer to the validation data (X_valid), transforming it into a matrix with the same dimensions as the training data.

```
[ ] # Vectorize the text data
    vectorizer = TfidfVectorizer()
    X_train_v = vectorizer.fit_transform(X_train)
    X_valid_v = vectorizer.transform(X_valid)
```

(3) Train and evaluate models

We fitted the training set into various classifiers, including Naive Bayes, SVM, Logistic Regression, Random Forest, and XGBoost. Then we used these classifiers to predict the validation set. This allowed us to evaluate and determine which classification model is the most suitable for our task.

We obtained an accuracy of 0.71 in Naive Bayes and Random Forest classifiers. The SVM and XGBoost classifiers achieved an accuracy of 0.75. Lastly, the Logistic Regression classifier performed the best with an accuracy of 0.76. Based on these results, it can be concluded that the Logistic Regression model is the most suitable for our classification task.

3. Train & Predict with Logistic Regression

(1) Feature extraction

After identifying the best classifier, we proceed to train the model once again using the entire dataset without splitting it into training and validation sets. Additionally, we convert the text data into a numerical representation using TF-IDF as before.

To accomplish this, vectorizer.fit_transform(train['content']) applies the vectorizer to the training data. Similarly, vectorizer.transform(test['content']) applies the same vectorizer to the testing data. This ensures that the testing data is transformed into a matrix with the same dimensions as the training data, allowing for consistent processing and compatibility with the trained model.

```
[ ] clf = LogisticRegression(max_iter = 1000, solver = 'saga')
    fit_evaluate(clf, X_train_v, y_train, X_valid_v, y_valid)
```

```
                precision    recall  f1-score   support

            -1       0.78      0.86      0.82      2106
             0       0.54      0.30      0.38      1280
             1       0.80      0.89      0.84      2986

      accuracy                           0.76      6372
     macro avg       0.70      0.68      0.68      6372
  weighted avg       0.74      0.76      0.74      6372
```

(2) Train & predict

Finally, we fit the training data into the Logistic Regression model and make predictions on the testing set.

```
[ ] clf = LogisticRegression(max_iter = 1000, solver = 'saga')
    clf.fit(train_X, train['category'])
```

```
LogisticRegression(max_iter=1000, solver='saga')
```

```
[ ] test['pred'] = clf.predict(test_X)
    test
```

	num	content	pred
0	1	outbound flight hour min flight thought sale s...	1
1	2	flight rhodes athens route heathrow cancelled ...	1
2	3	athens larnaca economy early morning flight de...	1
3	4	new new style seat quite uncomfortable though ...	1
4	5	flew london athens ioannina ioannina athens ba...	1
...	...	...	...
4995	4996	oct flt oz tired old plane minute late taking ...	−1
4996	4997	old plane old reclining seat poor entertainmen...	−1
4997	4998	flew recife brazil miami airline continues liv...	−1
4998	4999	travel internationally frequently business hol...	−1
4999	5000	part one world american airline codeshare mala...	1

5000 rows × 3 columns