Lab No: 5

Aim:
The aim is to employ Logistic Regression and Term Frequency-Inverse Document Frequency (TF-IDF) for spam classification, comparing accuracies across datasets to identify the most effective preprocessing technique.

Description:
Perform the following task with using inbuilt Python Libraries:

- Feature Extraction: Utilize TF-IDF vectorization to convert text data into numerical features.
- Data Splitting: Divide the datasets into 80% training and 20% testing subsets.
- Model Training: Train Logistic Regression models on the training data for each dataset.
- Prediction: Evaluate model performance by predicting labels on the testing sets.
- Accuracy Assessment: Calculate and compare accuracies to identify the most effective preprocessing technique among datasets.
- End Result: Determine which dataset, whether raw or preprocessed, yields the highest accuracy with Logistic Regression and TF-IDF.

Source Code:

```
# The aim is to employ Logistic Regression and Term Frequency-Inverse Document Frequency (TF-IDF) for spam classification, comparing accuracies across datasets to identify the most effective preprocessing technique.
```

```
## Perform the following task with using inbuilt Python Libraries: acy.
```

```
#### - Perform Classification of data sets (Data 1 (Raw Data), Data 2 (Data with Lowercase), ,..... Data n ) using Logistic Regression.
#### - Use Tf-Idf vectorizor for Feature Extraction.
#### - Use 80% of data for training and 20% of data for testing.
#### - Check the accuracy of the model for each dataset.
#### - Write conclusion for with data  (Data 1 (Raw Data), Data 2 (Data with Lowercase), ,..... Data n ), the Logistic Regression provides best accuracy.
```

```python
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import string
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
```

```python
data = pd.read_csv("spam.csv", encoding="ISO-8859-1")
data.head()
```

```python
data = data.drop(columns=["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"])
data.head()
```

```python
df = pd.DataFrame(data)
df.head()
```

```python
data_columns = ['v2', 'lowercased_v2', 'tokens', 'cleaned_v2', 'filtered_v2', 'stemmed_v2', 'lemmatized_v2']
accuracies = []
precisions = []
recalls = []
f1s = []
for column in data_columns:
    X = df[column].astype(str)
    y = df['v1']
    # Training
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    tfidf_vectorizer = TfidfVectorizer()
    X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
    X_test_tfidf = tfidf_vectorizer.transform(X_test)
    model = LogisticRegression()
    model.fit(X_train_tfidf, y_train)
    # Prediction
    y_pred = model.predict(X_test_tfidf)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label='spam')
    recall = recall_score(y_test, y_pred, pos_label='spam')
    f1 = f1_score(y_test, y_pred, pos_label='spam')
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f"\nResults for {column}:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print(f"Confusion Matrix:\n{conf_matrix}")
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
```

```python
    f1s.append(f1);
# Conclusion
best_index = accuracies.index(max(accuracies))
best_dataset = data_columns[best_index]
print(f'\nLogistic Regression provides the best accuracy with {best_dataset} having accuracy of {accuracies[best_index]}.')
best_index = precisions.index(max(precisions))
best_dataset = data_columns[best_index]
print(f'Logistic Regression provides the best precision with {best_dataset} having precision of {precisions[best_index]}.')
best_index = recalls.index(max(recalls))
best_dataset = data_columns[best_index]
print(f'Logistic Regression provides the best recall with {best_dataset} having recall of {recalls[best_index]}.')
best_index = f1s.index(max(f1s))
best_dataset = data_columns[best_index]
print(f'Logistic Regression provides the best F1 score with {best_dataset} having F1 score of {f1s[best_index]}.')
```

Output:

Spam Data:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Figure 5.0.1 Spam Data

After Dropping Unnecessary Columns:

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Figure 5.0.2 After Dropping Columns

After Applying Preprocessing Techniques:

	v1	v2	lowercased_v2	tokens	cleaned_v2	filtered_v2	stemmed_v2	lemmatized_v2
0	ham	Go until jurong point, crazy.. Available only ...	go until jurong point, crazy.. available only ...	[go, until, jurong, point, ,, crazy, .., avail...	gountiljurongpointcrazy..availableonlyinbugisn...	go jurong point , crazy .. available bugis n g...	go until jurong point , crazi .. avail onli in...	go until jurong point , crazy .. available onl...
1	ham	Ok lar... Joking wif u oni...	ok lar... joking wif u oni...	[ok, lar, ..., joking, wif, u, oni, ...]	oklar...jokingwifuoni...	ok lar ... joking wif u oni ...	ok lar ... joke wif u oni ...	ok lar ... joking wif u oni ...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	free entry in 2 a wkly comp to win fa cup fina...	[free, entry, in, 2, a, wkly, comp, to, win, f...	freeentryin2awklycomptowinfacupfinaltkts21stma...	free entry 2 wkly comp win fa cup final tkts 2...	free entri in 2 a wkli comp to win fa cup fina...	free entry in 2 a wkly comp to win fa cup fina...
3	ham	U dun say so early hor... U c already then say...	u dun say so early hor... u c already then say...	[u, dun, say, so, early, hor, ..., u, c, alrea...	udunsaysoearlyhor...ucalreadythensay...	u dun say early hor ... u c already say ...	u dun say so earli hor ... u c alreadi then sa...	u dun say so early hor ... u c already then sa...
4	ham	Nah I don't think he goes to usf, he lives aro...	nah i don't think he goes to usf, he lives aro...	[nah, i, do, n't, think, he, goes, to, usf, ,,...	nahidon'tthinkhegoestousfhelivesaroundherethough	nah n't think goes usf , lives around though	nah i do n't think he goe to usf , he live aro...	nah i do n't think he go to usf , he life arou...

Figure 5.0.3 After Applying Preprocessing Techniques

Output:

```
Results for v2:
Accuracy: 0.9659
Precision: 0.9912
Recall: 0.7533
F1 Score: 0.8561
Confusion Matrix:
[[964   1]
 [ 37 113]]

Results for lowercased_v2:
Accuracy: 0.9659
Precision: 0.9912
Recall: 0.7533
F1 Score: 0.8561
Confusion Matrix:
[[964   1]
 [ 37 113]]

Results for tokens:
Accuracy: 0.9686
Precision: 0.9915
Recall: 0.7733
F1 Score: 0.8689
Confusion Matrix:
[[964   1]
 [ 34 116]]
```

```
Results for cleaned_v2:
Accuracy: 0.8834
Precision: 1.0000
Recall: 0.1333
F1 Score: 0.2353
Confusion Matrix:
[[965   0]
 [130  20]]

Results for filtered_v2:
Accuracy: 0.9578
Precision: 0.9558
Recall: 0.7200
F1 Score: 0.8213
Confusion Matrix:
[[960   5]
 [ 42 108]]

Results for stemmed_v2:
Accuracy: 0.9668
Precision: 0.9829
Recall: 0.7667
F1 Score: 0.8614
Confusion Matrix:
[[963   2]
 [ 35 115]]
```

```
Results for lemmatized_v2:
Accuracy: 0.9677
Precision: 0.9914
Recall: 0.7667
F1 Score: 0.8647
Confusion Matrix:
[[964   1]
 [ 35 115]]
```

```
Logistic Regression provides the best accuracy with tokens having accuracy of 0.968609865470852.
Logistic Regression provides the best precision with cleaned_v2 having precision of 1.0.
Logistic Regression provides the best recall with tokens having recall of 0.7733333333333333.
Logistic Regression provides the best F1 score with tokens having F1 score of 0.8689138576779025.
```

Figure 5.1 Output

Conclusion:

- Applied preprocessing steps including lowercasing, tokenization, cleaning, filtering, stemming, and lemmatization to enhance text data quality.
- Utilized Tf-Idf vectorization for feature extraction, capturing term importance in each dataset.
- Trained Logistic Regression models on each preprocessed dataset using 80% of data for training and 20% for testing.
- Evaluated model accuracy for each dataset, measuring performance on spam classification.
- Identified the dataset with the highest accuracy, indicating that Logistic Regression performs best on token generation approach.