CISB5123 Text Analytics
Lab 6
Text Classification

In this lab, you will learn how to build a text classification model using a real-world dataset - the SMS Spam Collection. You will preprocess text data, convert it into numerical form using techniques like TF-IDF, and then train a classifier to distinguish between spam and ham (non-spam) messages.

Dataset

The dataset used is 'smsspamcollection.csv', which contains labeled SMS messages.

Load a Dataset

 # Perform imports and load the dataset: *import pandas as pd*
df = pd.read_csv("smsspamcollection.csv",encoding="ISO-8859-1") *df.shape*
df.head()
df.info()
 # Checking for null values *df.isnull().sum()*

```
# Checking Duplicate values
df.duplicated().sum()
```

```
# Drop Duplicate values
df=df.drop_duplicates()
df.shape
```

```
# renaming the columns
df.columns = ['label', 'message']
df.head()
```

Exploratory Data Analysis (EDA)

```
df.describe()
```

```
df.groupby('label').describe()
```

We have 4516 ham messages and 653 spam messages.

```
# convert label to a numerical variable
df['label_num'] = df.label.map({'ham':0, 'spam':1})
df.head()
```

```
df['message_len'] = df.message.apply(len)
df.head()
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 8))

df[df.label=='ham'].message_len.plot(bins=35, kind='hist', color='blue',
                                     label='Ham messages', alpha=0.6)
df[df.label=='spam'].message_len.plot(kind='hist', color='red',
                                      label='Spam messages', alpha=0.6)
plt.legend()
plt.xlabel("Message Length")
```

```
df[df.label=='ham'].describe()
```

```
df[df.label=='spam'].describe()
```

Text Pre-Processing

```
# import library
import re
import string
import nltk
from nltk.corpus import stopwords
```

```
stop_words = stopwords.words('english')
more_stopwords = ['u', 'im', 'c']
stop_words = stop_words + more_stopwords
```

```
stemmer = nltk.SnowballStemmer("english")
```

```
def preprocess(text):
    text = text.lower()  # Convert text to lowercase
    text = re.sub(r'\[.*?\]', '', text)  # Remove text within square brackets
    text = re.sub(r'http\S+\s*\S+', '', text)  # Remove URLs starting with http
    text = re.sub(r'www\.\S+', '', text)  # Remove URLs starting with www
    text = re.sub(r'<.*?>', '', text)  # Remove HTML tags
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\b\w*\d\w*\b', '', text)  # Remove words containing numbers
    text = ' '.join(word for word in text.split(' ') if word not in stop_words) #remove
stopwords
    text = ' '.join(stemmer.stem(word) for word in text.split(' ')) #stemming
    return text
```

```
df['message_clean'] = df['message'].apply(preprocess)
df.head(20)
```

```
print(df['message_clean'][304])
```

```
pip install wordcloud
```

```
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# Create a WordCloud object
wc = WordCloud(
    background_color='white',
max_words=200,
    contour_color='steelblue',  # Add contour color
    contour_width=2,  # Add contour width
)

# Generate the WordCloud using text data for HAM messages
ham_text = ' '.join(text for text in df.loc[df['label'] == 'ham', 'message_clean'])
wc.generate(ham_text)

# Visualize the WordCloud
plt.figure(figsize=(10, 8))
plt.imshow(wc, interpolation='bilinear')
plt.title('WordCloud for HAM messages', fontsize=20)
plt.axis('off')
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# Create a WordCloud object
wc = WordCloud(
    background_color='white',
    max_words=200,
    contour_color='steelblue',  # Add contour color
    contour_width=2,  # Add contour width
)

# Generate the WordCloud using text data for SPAM messages
ham_text = ' '.join(text for text in df.loc[df['label'] == 'spam', 'message_clean'])
wc.generate(ham_text)

# Visualize the WordCloud
plt.figure(figsize=(10, 8))
plt.imshow(wc, interpolation='bilinear')
plt.title('WordCloud for SPAM messages', fontsize=20)
plt.axis('off')
plt.show()
```

Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer()

X = tfidf_vect.fit_transform(df['message_clean'])
y = df['label']
```

Modeling

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

```
# Split data into train & test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
# Naive Bayes
nb_clf = MultinomialNB().fit(X_train, y_train)

# SVM
svm_clf = SVC(kernel='linear').fit(X_train, y_train)
```

Performance Evaluation

```
# Performance metrics comparison
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score  # Importing accuracy_score

# Naive Bayes
nb_predicted = nb_clf.predict(X_test)
nb_report = classification_report(y_test, nb_predicted)
# Calculate error rate for Naive Bayes
nb_error_rate = 1 - accuracy_score(y_test, nb_predicted)
nb_cm = confusion_matrix(y_test, nb_predicted)

# SVM
svm_predicted = svm_clf.predict(X_test)
svm_report = classification_report(y_test, svm_predicted)
# Calculate error rate for SVM
svm_error_rate = 1 - accuracy_score(y_test, svm_predicted)
svm_cm = confusion_matrix(y_test, svm_predicted)
```

```
nb_cm
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Function to plot confusion matrix
def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap=cmap, xticklabels=classes,
yticklabels=classes)
    plt.title(title)
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()

# Plot confusion matrix for Naive Bayes
plot_confusion_matrix(nb_cm, classes=['negative', 'positive'], title='Naive Bayes
Confusion Matrix')

print("Naive Bayes Classifier Report:")
print(nb_report)
print(f"\nNaive Bayes Error Rate: {nb_error_rate:.2f}")

# Plot confusion matrix for SVM
plot_confusion_matrix(svm_cm, classes=['negative', 'positive'], title='SVM Confusion
Matrix')

print("\nSVM Classifier Report:")
print(svm_report)
print(f"\nSVM Error Rate: {svm_error_rate:.2f}")
```

Conclusion

In this lab, you've implemented a basic text classification pipeline using TF-IDF and a machine learning classifier. You've also learned the importance of preprocessing and evaluating models using relevant metrics.

Exercise

You are provided with a file named 'Processed_Reviews.csv', which contains preprocessed reviews. Your task is to build a text classification model using the lemmatized column only.,

Instructions

1. **Manually Label the Data:**
    - Open the Processed_Reviews.csv file.
    - Add a new column named label.
    - For each review, assign a label:
        - For example: positive = 1, negative = 0.
    - Save the updated file.
2. **Load the Data in Python:**
3. **Pre-processing:**
    - Use only the lemmatized column.
    - Convert labels to numeric if they are categorical.
4. **Vectorization:**
    - Apply TfidfVectorizer to transform the text data into numerical features.
5. **Model Training:**
    - Split the data into training and test sets.
    - Train a classifier such as Naïve Bayes or SVM.
6. **Evaluation:**
    - Evaluate the model using accuracy, precision, recall, and F1-score.
    - Optionally, display the confusion matrix.