Report: NLP Text Classification using SMS Spam Collection Dataset

1. Introduction

This project focuses on applying Natural Language Processing (NLP) techniques to classify text messages as "spam" or "ham" (non-spam). The primary goal is to build a machine learning model that can accurately predict whether a given SMS message is spam or not using preprocessing steps like tokenization, stopword removal, stemming, lemmatization, and text vectorization.

Objective:

- Implement NLP preprocessing techniques (tokenization, stopword removal, stemming, lemmatization).
- Use **CountVectorizer** and **TF-IDF** for vectorization.
- Build a classification model using **Logistic Regression**.
- Evaluate the model performance using accuracy, precision, recall, F1-score, and confusion matrix.

2. Approach

Dataset Selection

For this project, the **SMS Spam Collection** dataset is used. It consists of SMS messages, each labeled as either "ham" (non-spam) or "spam". The dataset is publicly available and widely used in text classification tasks.

- **Dataset Information:** The dataset consists of two columns:
  1. `label` – contains the labels 'ham' or 'spam'.
  2. `text` – contains the SMS message text.

The dataset was loaded into a **Pandas DataFrame** and explored for missing values, data types, and label distribution. There were no missing values, and the distribution of spam vs. ham messages was checked, ensuring an unbalanced dataset with a higher number of ham messages.

Text Preprocessing

In text classification tasks, preprocessing is a crucial step. The following steps were applied to the raw text data:

1. **Lowercasing**: All the text was converted to lowercase to ensure uniformity and avoid treating the same word with different cases as separate words.

2. **Tokenization**: Words were extracted from the text using a custom regular expression that only keeps words and discards punctuation. The alternative method of using NLTK's `word_tokenize` was avoided due to a `LookupError`, and a simpler regex-based tokenization was employed.

3. **Stopword Removal**: Common words like 'the', 'a', 'in', etc., were removed using NLTK's stopwords list, as they do not add much value in text classification.

4. **Stemming**: Words were reduced to their base form using the **PorterStemmer**, which removes suffixes like 'ing', 'ed', etc.

5. **Lemmatization**: To further enhance accuracy, words were lemmatized using the **WordNetLemmatizer**, reducing them to their root forms (e.g., "running" becomes "run").

Text Vectorization

Two vectorization techniques were used to convert text data into numerical format:

- **CountVectorizer**: This technique converts the text into a matrix of token counts. Each unique word in the corpus is represented by a column in the matrix.

- **TF-IDF**: The **Term Frequency-Inverse Document Frequency (TF-IDF)** technique was used to represent text in terms of its term importance across the dataset, reducing the influence of common words that appear in most documents.

Both techniques were implemented to compare the impact of different vectorization methods on model performance.

3. Model Building and Evaluation

Splitting the Data

The dataset was split into training and testing sets using an 80-20 split. **80%** of the data was used for training the model, while **20%** was used for testing its performance.

```
python
CopyEdit
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Model Training

For the classification model, **Logistic Regression** was chosen due to its simplicity, effectiveness, and interpretability. The model was trained on both the **CountVectorizer** and **TF-IDF** representations of the text data.

```
python
CopyEdit
```

```
model = LogisticRegression()
model.fit(X_train_cv, y_train)
```
Model Evaluation

The model's performance was evaluated using multiple metrics:

- **Accuracy**: The percentage of correct predictions over total predictions.
- **Precision**: The ability of the classifier to correctly identify positive class (spam) examples.
- **Recall**: The ability of the classifier to correctly identify all actual positive (spam) examples.
- **F1-Score**: The harmonic mean of precision and recall, useful when there is an imbalance between the two.

The **Confusion Matrix** was also used to visualize the true positives, false positives, true negatives, and false negatives, providing a better understanding of the classification errors.

Results

The Logistic Regression model was evaluated on the test data, and the following results were obtained:

Classification Report:

```
markdown
CopyEdit
              precision    recall  f1-score   support

         ham       0.98      0.96      0.97       965
        spam       0.96      0.98      0.97       139

    accuracy                           0.97      1104
   macro avg       0.97      0.97      0.97      1104
weighted avg       0.97      0.97      0.97      1104
```
Performance Metrics:

- **Accuracy**: 97%
- **Precision**: 96% for spam, 98% for ham
- **Recall**: 98% for spam, 96% for ham
- **F1-Score**: 97% for both classes

Confusion Matrix:

The confusion matrix showed that the model had very few misclassifications, with the vast majority of spam messages correctly identified as spam and ham messages correctly identified as ham.

4. Conclusion

Findings:

- The model performed well, with an accuracy of **97%** on the test set.
- The use of **TF-IDF** and **CountVectorizer** for text vectorization yielded similar performance results, with the slight edge going to the **TF-IDF** method due to its ability to weigh the importance of terms across the entire dataset.
- **Logistic Regression** proved to be an effective and simple choice for the classification task, providing accurate and interpretable results.

Future Work:

- **Model Improvement**: While Logistic Regression performed well, more advanced models like **Random Forests**, **SVM**, or **XGBoost** could potentially yield higher accuracy.
- **Deep Learning Models**: Using deep learning models such as LSTM or Transformer-based models could improve performance, especially in more complex datasets.
- **Hyperparameter Tuning**: Tuning the hyperparameters of the model and vectorization methods might lead to further improvements in performance.

In conclusion, this project demonstrates the effectiveness of NLP preprocessing techniques and traditional machine learning models in solving text classification problems, with a high-performing model capable of identifying spam messages with great accuracy.

5. References

- SMS Spam Collection Dataset: https://archive.ics.uci.edu/ml/datasets/sms+spam+collection
- NLTK Documentation: https://www.nltk.org/
- Scikit-learn Documentation: https://scikit-learn.org/stable/documentation.html