NAME: **RAHUL BIRWADKAR**
MATRICULATION NO.: **11037364**
DATE OF SUBMISSION: **06/05/2024**

Task 2 : SMS Spam Detection.

Introduction

In this project, we are demonstrating the SMS spam detection. We are using Multinomial Naive bayes approach to tarin the model.

Implementation and Methodology

For the implementation of the project, VS code IDE is used, and Python programming language is used.

- Import the necessary libraries as follows:
    - Sklearn Feature Extraction, Naive Bayes, Model Selection.
    - Countvectorizer
    - MultinomialNB
    - Train test split
    - Accuracy and confusion matrix

- Data loading : Load the SMS Spam Collection dataset.
- Data Splitting : Split the dataset, 80% training data and 20% testing data.
- N-gram : Define the Character N-gram vectorizer . Here I am using bi-gram and trigram.
- Model Training : Multinomial Naïve Bayes classifier is used.
- We calculate the confusion matrix to evaluate the performance of the classification model.
- The confusion matrix provides a breakdown of the number of true positives, true negatives, false positives, and false negatives.
- Using the values from the confusion matrix, we calculate precision, recall, and F1 score.

Results:

```
Model Accuracy: 0.9829596412556054

Confusion Matrix:
[[957   9]
 [ 10 139]]
Precision: 0.9391891891891891
Recall: 0.9328859060402684
F1 Score: 0.936026936026936
```

Figure 1: Output

Questions

- ➢ What is the order of your n-gram models (e.g., bigram models, trigram models, etc.)?
- – In the provided code, the order of the n-gram models is defined by the ngram_range parameter of the CountVectorizer.
- – Specifically, the code uses character n-grams ranging from 2 to 3 characters (ngram_range=(2, 3)). This means the code considers both bigrams, trigrams.

- ➢ How do you define and handle out-of-vocabulary symbols?
- – Out-of-vocabulary symbols refer to characters or character sequences that are not present in the vocabulary learned from the training data.
- – In the provided code, out-of-vocabulary symbols are handled implicitly by the CountVectorizer and MultinomialNB implementations in scikit-learn.
- – The CountVectorizer ignores unseen characters or character sequences during transformation, while the MultinomialNB model applies smoothing techniques to handle unseen n-grams during training and prediction.

- ➢ Which smoothing method do you apply? How do you avoid zero probabilities?
- – The Multinomial Naive Bayes model implemented in scikit-learn applies Laplace (add-one) smoothing by default.
- – This smoothing method adds a small constant (usually 1) to the count of each feature (n-gram) in every class.
- – This prevents zero probabilities and helps avoid overfitting by assigning non-zero probabilities to unseen features.

- ➢ Do you consider n-gram models of different orders? If so, how does the n-gram order affect the classification accuracy?
- – Yes, the code considers n-gram models of different orders by specifying the ngram_range parameter in the CountVectorizer.
- – By including a range of n-gram sizes (from 2 to 3 in this case), the code accounts for different levels of linguistic complexity and captures varying degrees of contextual information from the text data.
- – Generally, higher-order n-grams can capture more nuanced patterns but may also increase the dimensionality of the feature space and the risk of overfitting.
- – Therefore, it's essential to experiment with different n-gram orders to find the optimal balance between accuracy and model complexity.

References:

[1] Build a machine learning email spam detector with Python - LogRocket Blog
[2] Multinomial Naive Bayes - GeeksforGeeks
[3] gnjatovic.info: The Milan Gnjatovic Website