SPAM or HAM

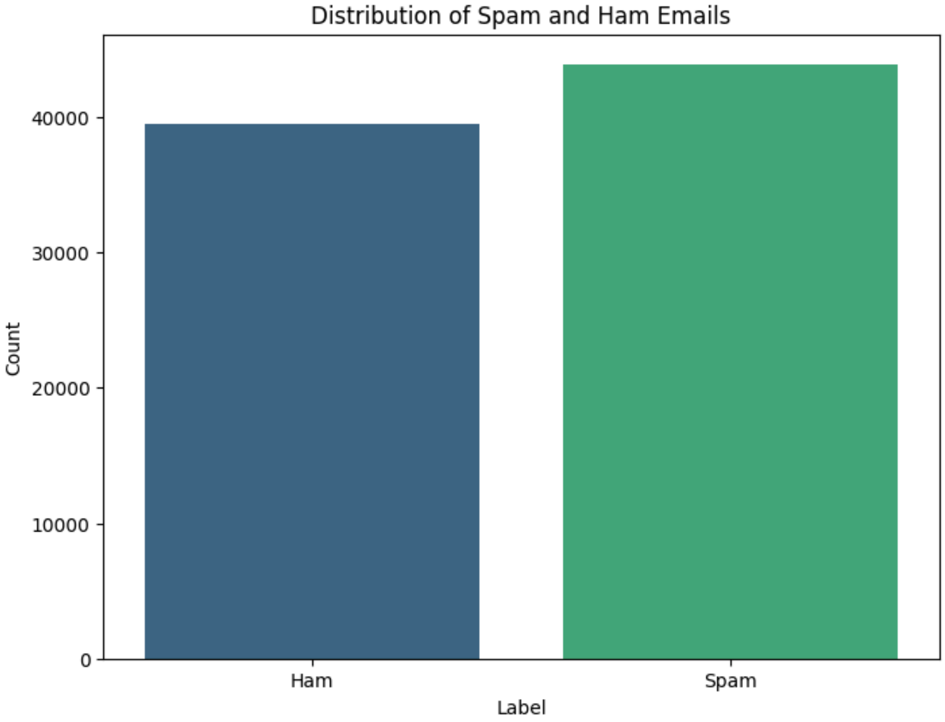
1. Introduction

The purpose of this assignment is to develop a spam classifier from scratch. I have used the Naive Bayes algorithm. The classifier distinguishes between **spam** and **ham** based on patterns in the email content. It involves preprocessing raw email data, extracting meaningful features, implementing a custom Naive Bayes classifier, and evaluating the model's performance.

2. Dataset

Dataset Source: Dataset Source Link
It includes labeled email data, marked as either "spam" or "ham" (non-spam), enabling us to train a binary classifier.

Dataset Structure:
File Format: **CSV**
Shape of the dataset:(83448,2)



This distribution shows that I have picked the **balanced dataset**. Spam and Ham dataset are equally distributed.

3. Feature Extraction

To distinguish between spam and ham emails, specific features were extracted from each email. These features were chosen based on patterns commonly found in spam messages.

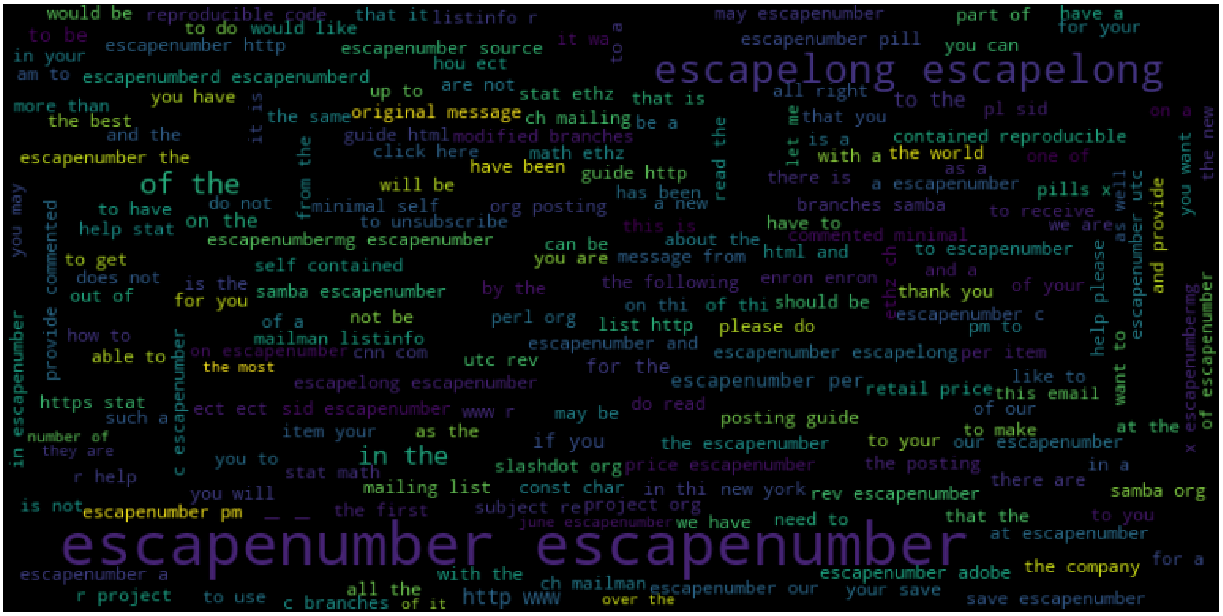
Extracted Features:

1. Word Frequency: The occurrence of each word in an email. Word counts help identify keywords that frequently appear in spam emails (e.g., "win", "free", "click").

2. Number of Links: The count of hyperlinks (e.g., `http://` or `https://`). Many spam emails include links to external websites.
3. Special Characters: The frequency of characters like `$`, `!`, `%`, `&` that are commonly used in spam for emphasis or to bypass filters.
4. Total Word Count: The overall number of words in each email, which can help differentiate between brief legitimate messages and longer spam messages.

These features are used to calculate the likelihood of an email being spam or ham by computing probabilities for each word and pattern.

Word Cloud of Email Content



4. Algorithm and Implementation

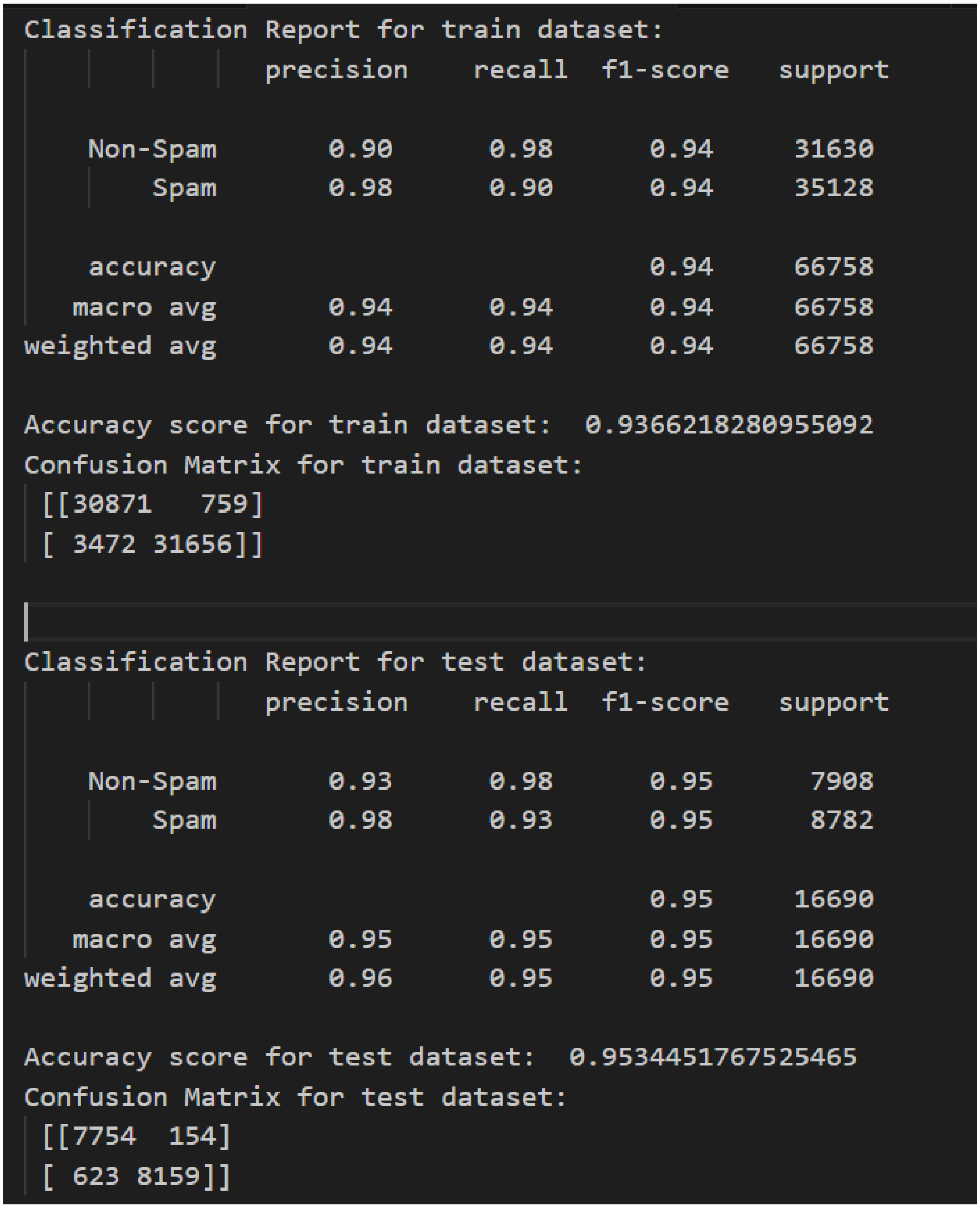
A Naive Bayes classifier was selected for this task due to its simplicity, interpretability, and efficiency in handling text data. The Naive Bayes classifier operates on the assumption that features (words, links, special characters) are conditionally independent given the label (spam or ham), which simplifies probability calculations.

80% of the dataset is used for training and 20% of the dataset is used for testing purposes. Dataset are divided in such a way that labels are in the **same proportion** train and test dataset using stratify method in the train_test_split.

Why Naive Bayes?
Efficiency: The Naive Bayes algorithm is computationally efficient, making it well-suited for real-time spam classification.
Text Classification Suitability: Naive Bayes is widely used in text classification tasks, such as spam detection, because of its effectiveness with high-dimensional data (large vocabularies).

Implementation Steps:
1. Probability Calculation: Calculate the probability of each word occurring in spam and ham emails separately, applying **Laplace smoothing** to handle words that may not appear in the training data.
2. Training the Model: Compute the probability distributions for spam and ham by analyzing word frequencies in each class.
3. Prediction: For each email, calculate the log-probabilities of it being spam or ham based on the extracted features, then classify it based on the higher probability.

5. Observations



```
Classification Report for train dataset:
               precision    recall  f1-score   support

    Non-Spam       0.90      0.98      0.94     31630
        Spam       0.98      0.90      0.94     35128

    accuracy                           0.94     66758
   macro avg       0.94      0.94      0.94     66758
weighted avg       0.94      0.94      0.94     66758

Accuracy score for train dataset:  0.9366218280955092
Confusion Matrix for train dataset:
 [[30871   759]
 [ 3472 31656]]


Classification Report for test dataset:
               precision    recall  f1-score   support

    Non-Spam       0.93      0.98      0.95      7908
        Spam       0.98      0.93      0.95      8782

    accuracy                           0.95     16690
   macro avg       0.95      0.95      0.95     16690
weighted avg       0.96      0.95      0.95     16690

Accuracy score for test dataset:  0.9534451767525465
Confusion Matrix for test dataset:
 [[7754  154]
 [ 623 8159]]
```