DA5400 Foundations of Machine Learning

Assignment 2
Spam / Ham Classifier

Roll no. - DA24C026
Name - Rishabh Gag

Problem Statement

Develop a spam classifier from scratch, utilising machine learning to distinguish between spam and non-spam emails.

Approach

The process of development of this spam classifier was structured file by file for clear organisation and functionality.

The submission contains following scripts :-

1. **data_preparation.py**: This script combined and cleaned multiple datasets, standardising labels and balancing spam and ham samples. It resulted in a consolidated dataset, final_data.csv, ready for analysis and model training.
2. **EDA.py**: This file handled exploratory data analysis, calculating metrics like email length, word count, and sentence count. Visualisations and correlation matrices helped uncover patterns that guided feature selection.
3. **Spam_classifier.py**: This file implemented the classifier using a Naive Bayes model implemented from scratch. After vectorizing emails with a binary bag-of-words approach, the model was trained and evaluated on four test samples, outputting classification metrics and confusion matrices.
4. **Main.py**: The final script automated the testing process, reading raw email.txt files, classifying each as spam or ham, and saving predictions to predictions.csv

The Naive Bayes model was chosen for this assignment due to its effectiveness as a *generative* model in high-dimensional, text-based applications like spam detection. Its *probabilistic* approach allows it to calculate the likelihood of each class (spam or ham) based on *word occurrences*, making it highly interpretable and efficient.

This report covers each phase in detail, highlighting the methodology, insights gained from EDA, and the classifier's practical performance.

I. Data Preparation (`data_preparation.py`)

The final dataset, `final_data.csv`, was created by integrating and cleaning multiple datasets. The steps included label standardisation, sampling, and balancing to create a dataset ideal for training a binary classifier.

Sources of Data

- The main dataset sources included `llm_gen_emails.csv`, `s1.csv`, `s2.csv`, `s3.csv`, and `s4.csv`.
- Each file contained mixed spam and ham emails with varying labelling conventions.

Steps in Data Preparation

- Column Filtering: Only relevant columns (`email` and `label`) were retained.
- Label Standardisation: Labels were standardised to binary values, with `spam` mapped to `1` and `ham` to `0`.
- Sampling:
    - In `s1.csv`, 1000 ham emails were randomly sampled to balance with existing spam samples.
    - In `s2.csv`, 500 spam and 250 ham emails were sampled to create a balanced subset.
    - In `s3.csv`, "not spam" was mapped to ham (`0`).
    - In `s4.csv`, 1000 spam and 1000 ham emails were sampled.
- Merging and Shuffling: These cleaned datasets were merged and shuffled to create a single, balanced dataset.

Here’s the distribution of labels in the final dataset :-

Label	Count
Spam	2862
Ham	2889

II. Exploratory Data Analysis (`EDA.py`)

EDA was performed on `final_data.csv` to explore spam and ham characteristics, supporting feature selection for the model.

Feature Calculation

Three primary features were calculated for each email:

- Email Length: Total number of characters in each email, capturing the density of content.
- Word Count: Number of words in each email, reflecting the volume of information.
- Sentence Count: Number of sentences, providing insights into structure and verbosity.

Insights and Observations

Average Length, Word Count, and Sentence Count:

- Spam emails generally have slightly fewer words and sentences than ham emails, suggesting a more concise format. The reduced sentence count and word length may indicate simpler language or repetitive phrases typical in spam content.

- Correlation Matrix: A high correlation was observed between `mail_len`, `num_words`, and `num_sent`, which validated that these features are interrelated and could collectively provide informative patterns for distinguishing spam from ham.

EDA Output:

Average Length of Mails:

Label	Length
Spam	750.480084
Ham	786.182416

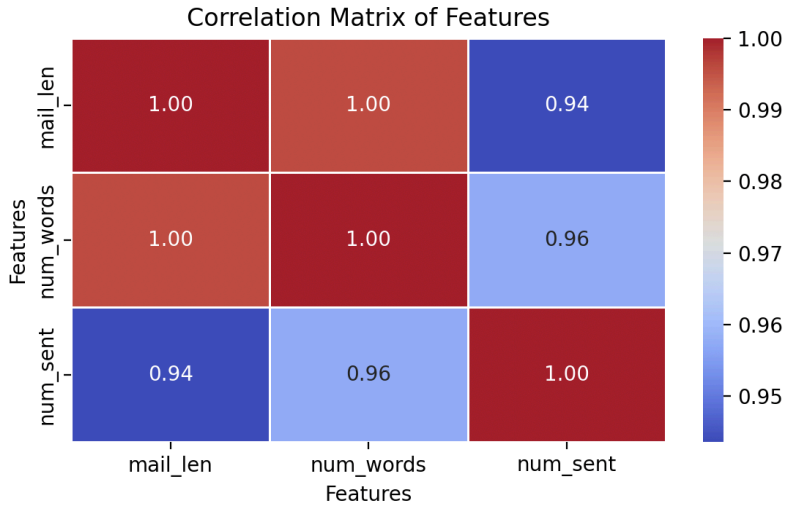
Average number of words in Mails:

Label	Words
Spam	140.103774
Ham	158.034960

Average number of sentences in Mails:

Label	Sentences
Spam	7.516422
Ham	8.355486

Correlation matrix:



Conclusion from EDA:

EDA revealed that spam emails are often shorter and simpler than ham emails. This characteristic is effectively captured by the binary bag-of-words approach, which the Naive Bayes model uses to detect patterns of word presence in each class. The insights gained through EDA helped in validating the feature engineering and model choice.

III. Naive Bayes Classifier Implementation (`Spam_classifier.py`)

The Naive Bayes classifier was implemented to classify emails based on their text features. This section explains how Naive Bayes was implemented from scratch.

Further the model was analysed on different test samples to ensure it serves the purpose of our classifier.

Theory and Assumptions of Naive Bayes:

Naive Bayes classifiers operate under the assumption of conditional independence between words, meaning each word's contribution to spam or ham classification is independent of other words. This assumption aligns well with text classification where a *word's presence or absence* can strongly indicate a class, especially in high-dimensional spaces.

Code Structure and Implementation:

- Initialization of CountVectorizer: We began by initialising the countvectorizer, setting the binary parameter to True. This configuration means that the vectorizer will create a binary occurrence matrix where each word's presence in an email is marked with a 1, and its absence is marked with a 0.

  The output is a sparse binary matrix where rows correspond to individual emails, columns correspond to unique words (features) extracted from the entire corpus and values indicate the presence (1) or absence (0) of each word in the respective email.

- Prior Probabilities:

  The priors for spam and ham are computed based on their proportions in the training dataset, allowing the classifier to weight each class by its prevalence.

- Word Probabilities with Laplace Smoothing:

  Laplace smoothing was applied to avoid zero probabilities for words not seen in a particular class. The vocabulary size was added to the denominator, ensuring the model remains robust against unseen words.

- Spam and Ham Word Probabilities: These were calculated by checking occurrences of each word across spam and ham emails, divided by the total word counts for each class.

- Prediction: For each email, the classifier computes the log-likelihood for spam and ham classes based on word presence. This is achieved by summing the log probabilities, which allows a direct comparison of spam vs. ham likelihoods.

  Calculation of Log likelihood for a class C is given by:

$$\log P(C|X) = \left(\sum_{i=1}^{n} \log P(x_i|C) \right) + \log P(C)$$

Why Naive Bayes Works Well Here

- Efficient Computation: Naive Bayes is computationally efficient, especially suitable for high-dimensional feature spaces such as text data.

- Effectiveness in Text Classification: The Naive Bayes model excels in text classification tasks because of strong indicative patterns of word presence, which is common in spam detection.

- Alignment with EDA Findings: EDA showed that certain patterns, such as fewer sentences in spam, align with Naive Bayes's ability to classify based on distinct word and structure patterns.

Naive bayes Model Evaluation

The classifier was tested on four randomly sampled test datasets (downloaded from different sources), each containing 2,500 emails. The evaluation metrics used were precision, recall and F1-score.

Performance Analysis

The classifier's performance on each test sample was as follows:

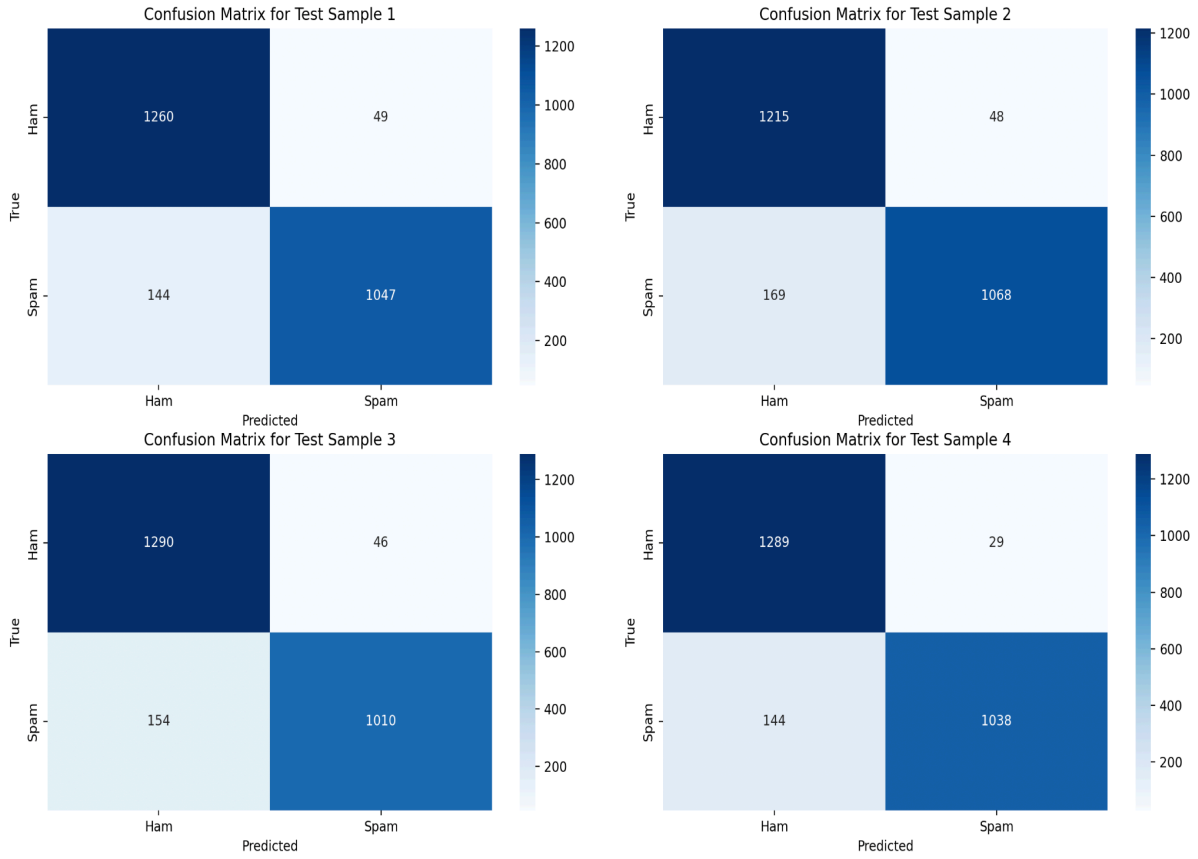
- Precision and Recall: The model showed high precision and recall for both spam and ham, indicating its ability to accurately classify both classes.
- F1 Score: The classifier achieved consistent F1 scores across all samples, with scores ranging from 91% to 93%.

Output on test samples:

Test Sample	F1 Score
I	0.92
II	0.91
III	0.92
IV	0.93

The confusion matrices for each test sample provided further insights into the model's performance, particularly regarding false positives and false negatives.

Confusion Matrices:



Confusion Matrix Observations:

- High true positive and true negative rates across all samples.
- Few false positives and negatives, showing the classifier's strong generalisation ability.

Interpretation of Results

The model's performance highlights its effectiveness in distinguishing between spam and ham emails. High precision and recall values indicate that the model is accurate in flagging spam emails, while the *balanced* sampling in training helped *minimise* bias.

IV. Testing and Prediction Automation (`main.py`)

The `main.py` file automates the prediction process, demonstrating the classifier's usability in a prediction process.

This script processes `.txt` email files from a designated test directory:

- Each `.txt` file is read and stored as a structured DataFrame with columns for the file name and email content.
- The trained `CountVectorizer` is applied to transform the emails into binary word presence vectors, matching the training format.

The classifier's `predict` method assigns a label to each email as either spam (`1`) or ham (`0`).

The predictions are saved in `predictions.csv`, providing a complete log of predictions. This CSV contains all 0's and 1's predictions with header *'Predictions'*.

Conclusion

In this assignment we successfully implemented a Naive Bayes spam classifier from scratch with a structured workflow for data preparation, EDA, model training, and deployment. The classifier demonstrated robust accuracy and efficiency across multiple test samples, confirming the suitability of Naive Bayes for spam detection.