Naive Bayes Classification Experiment

Tian Tian

Student ID: 2021011048

I. Implementation and Evaluation of Naive Bayes Classifier

1. Experiment Design

First, we need to convert the entire content of each email into features that can be processed. During this process, we need to choose which information to use and what model to use to convert it into features. Then, we train the Naive Bayes classifier model using the extracted features and classification results. Finally, we evaluate the performance of the model using custom evaluation metrics.

2. Feature Transformation

(1) Selecting Email Content

The emails we need to process are all in English and are divided into two parts: email-related information and email content.

The email-related information includes the sending time, sender's IP address, email ID, sender and recipient's email addresses, email subject, email priority, email client information, and email content type and encoding method. Manual observation revealed that not all emails contain all the email-related information. To specifically determine which email headers are included in the dataset, we used Python's built-in email library to traverse all the email headers. We found a total of 782 email headers. Based on personal judgment and classroom discussions, we finally selected the following email headers for feature transformation:

```
From: Sender's email, which can potentially indicate spam if the suffix suggests bulk registration. If the suffix is
.edu, it's likely not spam.
Cc: Carbon copy, which indicates that if this appears, it is likely not spam.
Bcc: Blind carbon copy, similar reasoning as Cc.
Subject: Subject, which can help determine if it is spam based on the content.
Date: Emails sent late at night are more likely to be spam.
Content-Type: The type and encoding of the content, where HTML encoded emails are more likely to be spam.
Content-Length: The length of the content, where spam emails might be longer.
Content-Transfer-Encoding: Defines how the email is encoded.
X-Authenticated, X-Authentication, X-Auth: Whether the email is authenticated, with authenticated emails being less
likely to be spam.
X-Priority, X-MSMail-Priority: The priority of the email, with marked priority emails generally not being spam.
```

The email content consists of two parts: plain text and HTML information. Emails may contain both parts or just one part. During training, the entire content of the email is used for feature transformation.

(2) Selection of Transformation Model

Considering both the simplicity of implementation and the effectiveness of the model, we use the Bag-of-Words (BoW) model for feature transformation. In section three, we will compare and discuss the impact of using different feature transformation methods on the final performance.

Based on classroom discussions and my own thoughts, I believe that email header information is particularly unique compared to email body content: headers are shorter and potentially more representative. For example, if the header indicates that the email was carbon copied (Cc) or blind carbon copied (Bcc), it's likely not spam. Therefore, we shouldn't mix header and body content for word frequency statistics. Many contents in headers might not be important themselves, but their presence is, such as the three indicators of authentication. Thus, it's necessary to extract features from the headers separately.

Based on my analysis and feature extraction techniques, I plan the following feature extraction method.

For the email body, we choose the BoW model and use chi-square testing to select the number of features for word frequency statistics.

The principle of chi-square testing is to measure the correlation between the word frequency and the email category, selecting words with high correlation as features. The reason for choosing chi-square testing is initially planning to include all word frequencies, but the data volume was too large for my computer's memory, and clearly not all word frequencies are important, as most words appear only a few times. Hence, I plan to convert only a portion of word frequencies into features.

However, we cannot simply select words based on their frequency. For instance, only a small portion of spam emails might be product promotions, hence words related to promotions might have low frequency but are crucial for spam detection. Therefore, I use chi-square testing to select words, based on the correlation between their frequency and email classification. The number of words selected by chi-square testing is a hyperparameter.

For email headers, as they are shorter but crucial, we extract features separately without combining them with the body content.

First, we perform word frequency statistics on headers using chi-square testing, creating a dedicated feature vector for header information.

Next, we binary encode header information, recording its presence or absence to aid spam detection based on the existence of certain headers (Cc, Bcc, etc.).

Considering the importance of the email subject, we may separately analyze word frequency and content for subjects and sender information to distinguish educational emails with suffixes like .edu. If the original model performs poorly, this method can be used.

3. Training Process

I chose the Naive Multinomial Bayes Network implementation.

Initially, I selected 300 header features and 2000 body features, resulting in severe overfitting. The accuracy on the training set was 0.9005 and the precision was 0.9842, indicating good performance; however, the accuracy on the test set was only 0.35, worse than random guessing. Reducing feature quantity improved the results but still didn't exceed 50% accuracy.

Through online research, I realized the need for text cleaning before training, removing HTML content, extra whitespace, and stop words. Since emails containing URLs are likely spam but URL content is insignificant, I replaced all URLs with "http".

Parameter tuning showed that increasing the number of words for feature calculation improved training set performance but reduced test set performance, indicating overfitting. Therefore, I modified the model structure by adding Laplace smoothing to avoid zero probability issues mentioned in section three, reducing overfitting with increased total word count.

Moreover, I found that inspecting selected feature words helped verify if the model operated as expected, aiding model adjustment. Without chi-square testing, words selected by the BoW model were often meaningless, derived from a long garbled email, and performed poorly. I also noticed frequent appearance of numbers like 2022, 111, unrelated to spam. Thus, I removed numbers from text during preprocessing.

My final chosen parameters are: 21 header feature words, 40 body feature words, with Laplace smoothing alpha set to 100. Specific performance analysis is in the next section.

Using five-fold cross-validation, feature words are selected for each fold separately. For brevity, I merged all selected feature words from the five trainings in the report. Detailed results for each training can be found in `hw1.ipynb`.

Alphabetically sorted selected header feature words:

```
['_dragon', 'alternative', 'ascii', 'boundary', 'charset', 'dmdx', 'edt', 'est', 'flowed', 'format', 'forster',
'handyboard', 'html', 'iso', 'jonathan', 'jp', 'multipart', 'normal', 'paper', 'plain', 'text']
```

Alphabetically sorted selected body feature words:

```
['arizona', 'ascii', 'board', 'code', 'com', 'content', 'crust', 'data', 'date', 'dec', 'digest', 'dmdx', 'edu',
'esmtp', 'file', 'files', 'gt', 'id', 'kb', 'know', 'list', 'lt', 'mail', 'message', 'network', 'nil', 'owner',
'padding', 'pnfs', 'price', 'problem', 'product_table', 'psy', 'psych', 'px', 'received', 'reply', 'rpcss', 'send',
'sender', 'set', 'subject', 'telecom', 'thanks', 'thu', 'type', 'ucsb', 'university', 'use', 'using', 've', 'version',
'vulnerable', 'wed', 'wrote']
```

4. Performance Evaluation

From a qualitative perspective, I believe that the special aspect of spam classification tasks is that false negatives can be tolerated, but classifying legitimate emails as spam can have more severe consequences.

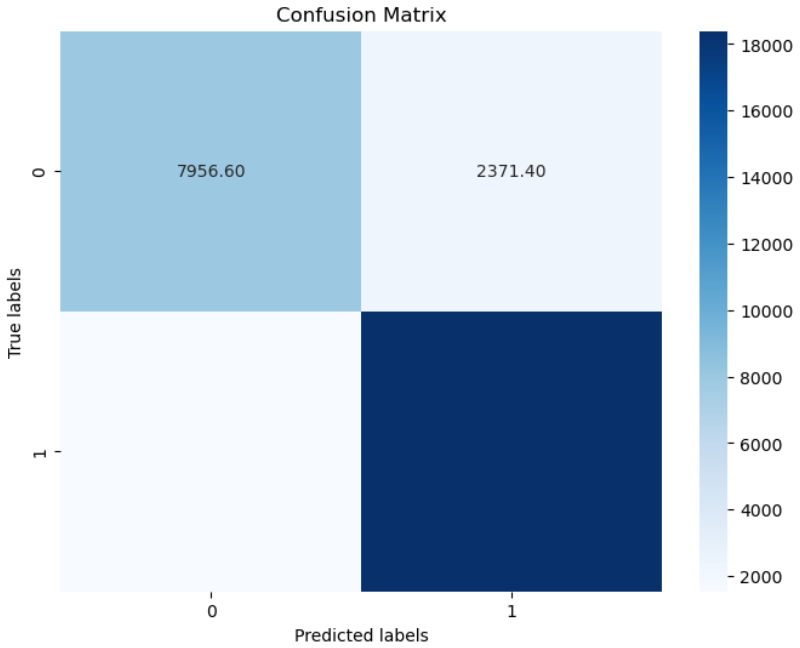
Therefore, we use accuracy to measure the overall success rate of the model's classification, and we use precision, recall, and F1 score to specifically assess the model's ability to detect spam. We will plot a heatmap of the confusion matrix to observe the classifier's performance in classification results comprehensively. Since spam emails are twice as frequent as non-spam emails in the training dataset, with an imbalanced number of positive and negative examples, we use the PR curve to examine the trade-off between precision and recall and use the Kappa coefficient to measure classification consistency.

Since five-fold cross-validation is used, all metrics mentioned below are the averages of the results calculated for each fold.

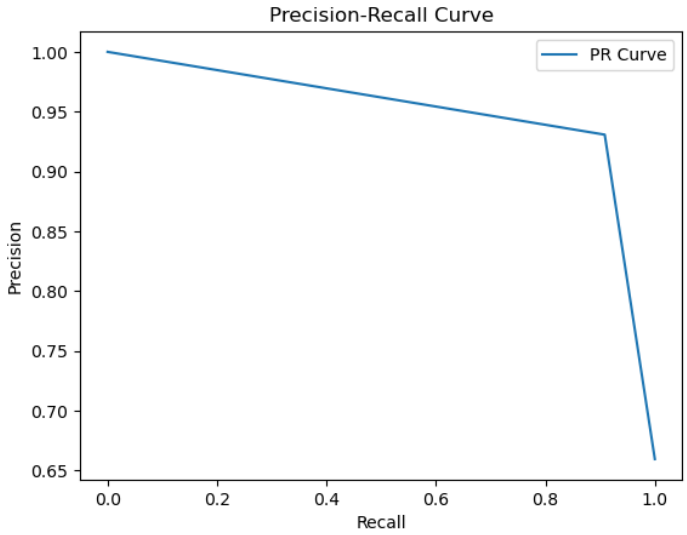
(1) Performance on the Training Set

On the training set, the model has relatively good performance. The accuracy is 0.871, the precision is 0.887, the recall is 0.923, and the F1 score is 0.904, indicating a good balance between precision and recall.

Since the results of five-fold cross-validation are the averages of the five folds, the quantities of the four categories in the confusion matrix are not integers. This is not due to calculation errors. Overall, errors are mainly concentrated on classifying legitimate emails as spam, which does not align with our desired outcome.



The PR curve mainly focuses on the precision performance at different recall rates, analyzing the balance between precision and recall in the model. The larger the area under the PR curve, the better the model's performance in identifying spam. Overall, the area under the PR curve is relatively large, indicating good performance on the training set.

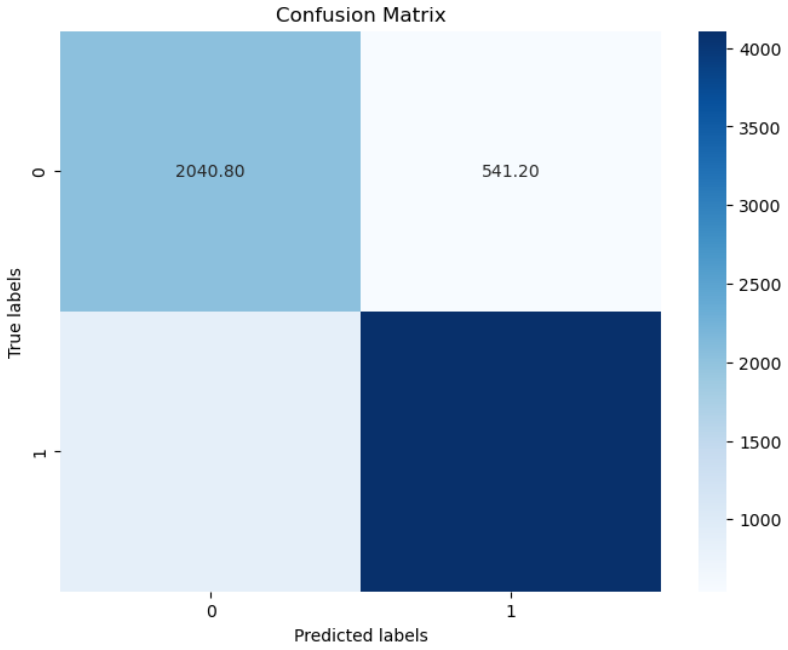


The Kappa statistic mainly measures the difference between the accuracy of the classification prediction and random prediction, used to evaluate the consistency of the classification ability. It is generally considered that the model's classification ability is good when the Kappa statistic is greater than 0.7. On the training set, the Kappa statistic is 0.7065, indicating good consistency performance on the training set.

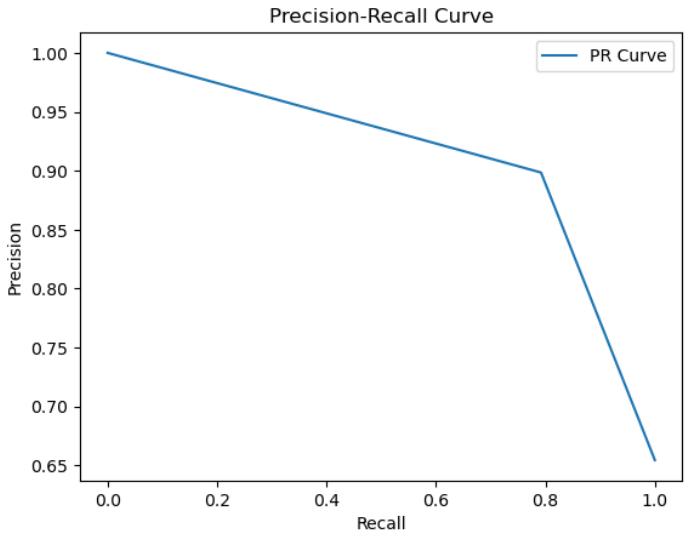
(2) Performance on the Test Set

On the test set, the model has a significant gap in accuracy compared to the training set. The accuracy is 0.813, the precision is 0.885, the recall is 0.8246, and the F1 score is 0.852, indicating good judgment ability and balance between precision and recall.

Overall, the confusion matrix shows that errors are mainly concentrated on classifying spam emails as legitimate emails, which is an acceptable type of error, performing better than on the training set.



Overall, the area under the PR curve is also relatively large, indicating good performance on the test set.



On the test set, the Kappa statistic is 0.5965, showing a significant decline in consistency performance compared to the training set.

Overall, despite reducing the model size to avoid overfitting and introducing Laplace smoothing to avoid zero probability issues mentioned in section three, overfitting remains a problem, and the model's final performance is not ideal.

5. Result Analysis

(1) Addressing the Required Questions

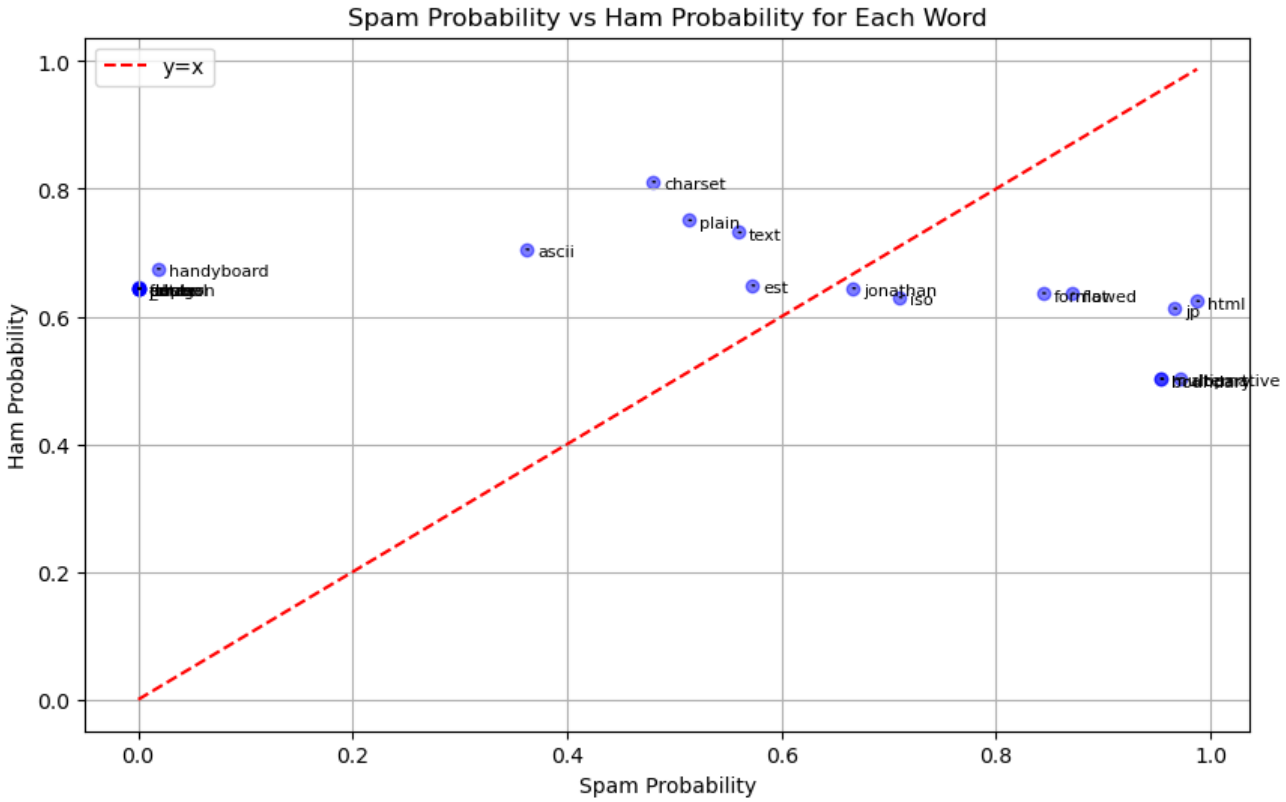
- **What problems did you encounter or explore in the experiment?**
  - The main issues encountered during this experiment were focused on the training process, primarily manifesting as poor model performance.
  - Specifically, the reasons for the poor model performance that I could identify include:
    1. Emails may contain long HTML-formatted text, which interferes with word selection in the Bag-of-Words model.
    2. If words are selected based solely on frequency, many selected words might be meaningless, possibly recurring content from a long email.
    3. Words selected using chi-square testing in the Bag-of-Words model often appear as number strings, which seem meaningless.
    4. Severe overfitting issues when a large number of features are selected.
- **How did you design further experiments upon encountering problems?**

    - Upon encountering problems, I first analyzed the causes by carefully reading the code, reviewing the experimental logic, and researching others' experiences with similar issues. Then, I attempted to introduce new techniques or modify parameters to improve the model's performance.
- **How did you adjust the algorithm to address the problems?**
    - First, understanding the root cause of the issue from the fundamental logic level is necessary.
    - Then, identifying relevant techniques to solve the problem by researching others' experiences online.
    - Specific adjustments to the algorithm included:
        1. Performing text cleaning before feature transformation, removing HTML content and other irrelevant information (details are in the training process section).
        2. Introducing chi-square testing to select words with high correlation to spam.
        3. Removing numbers during text cleaning.
        4. Applying Laplace smoothing to avoid zero probability issues, repeatedly tuning parameters to reduce model size, and checking selected feature words to ensure they are logical and effective. However, this problem was not fundamentally solved.
- **Did the adjustments improve the model's effectiveness?**
    - Overall, the adjustments successfully improved the model's effectiveness.
    - For issues other than overfitting, the adjustment methods were relatively effective.
    - However, I could not completely solve the overfitting problem, only managing to slightly reduce its impact.

(2) Further Analysis of Results

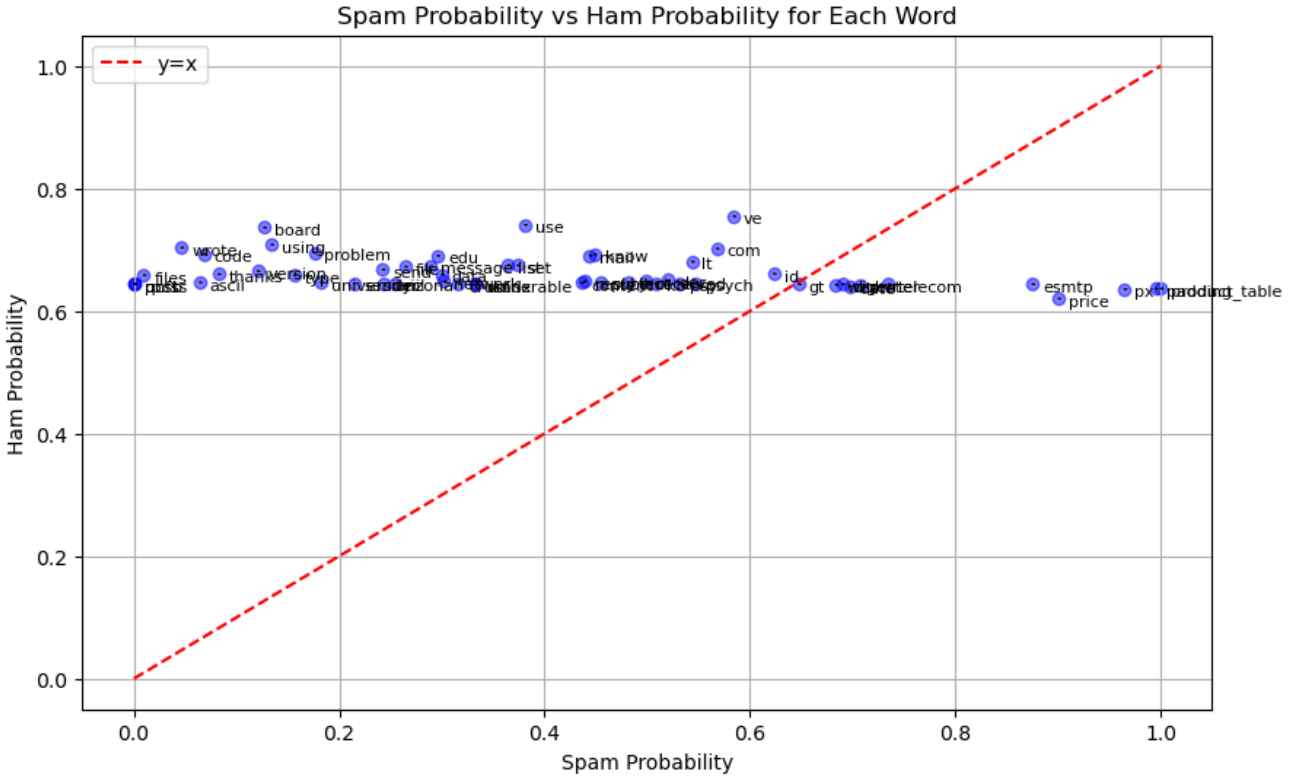
I found that the extracted keywords looked strange, with many words seemingly unrelated to whether an email is spam. Therefore, I suspect errors might have been included in the final content. Thus, I aim to specifically examine the relationship between feature words and spam, mainly using training data.

For the extracted feature words in email headers, using all the data, I plotted a scatter plot. The horizontal axis represents the frequency of spam emails containing the feature words, while the vertical axis represents the probability of non-spam emails without the feature words.



The plot shows that most selected feature words are far from the $y = x$ line, indicating clear irrelevance. However, many selected words, such as `est`, have no apparent meaning or correlation with spam, raising questions about why the algorithm selected them.

Performing the same analysis on the feature words in the email body produced a similar plot.



The analysis results are similar, with many feature words close to the $y = x$ line, questioning their selection.

However, we might rethink our approach based on this. From the currently selected words, we can reselect words far from the $y = x$ line for feature extraction and then retrain the model to observe if performance improves.

Following this idea, we retrained the model. For each split, we first extracted feature words using the original method and parameters. Then, we calculated the distance of these feature words from the $y = x$ line, selecting the words farthest from the line as final feature words for feature vector extraction and model training.

In this algorithm design, in addition to the original three hyperparameters, a new hyperparameter is added for reselecting words from the initially extracted feature words. After tuning, the final method was to select the three-fifths of the words farthest from the $y = x$ line and discard the five farthest points to avoid zero probability issues mentioned in problem two.

The final results on the training set were similar to the original model:

	Accuracy	Precision	Recall	F1 Score	Kappa
New Model	0.8203	0.8403	0.9031	0.8692	0.5832
Original Model	0.8129	0.8848	0.8246	0.8529	0.5965

Since I came up with this idea on the last night, there was not much time to specifically debug and optimize the implementation. I could only hurriedly implement the basic idea. However, I believe that a model constructed using this method, with careful parameter tuning, could certainly achieve better results than the original model.

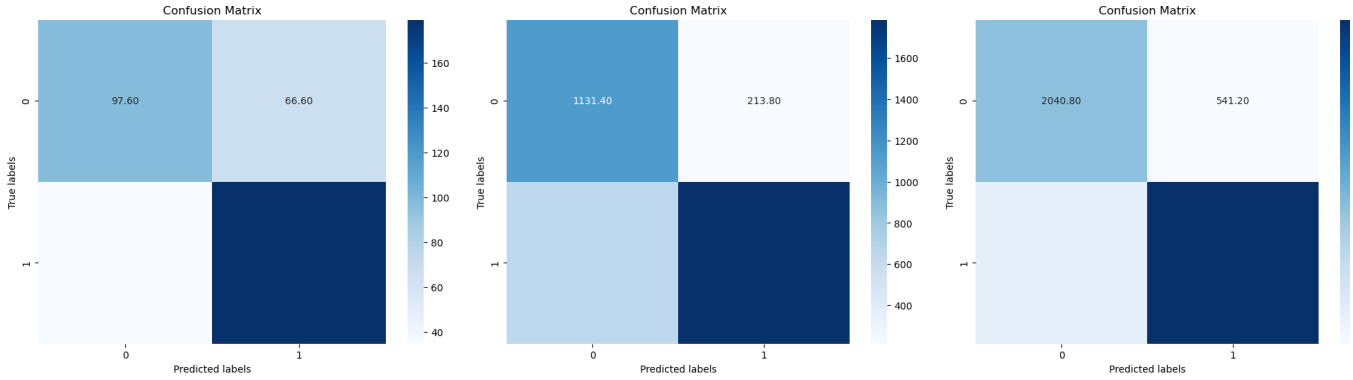
II. Answer to Question One

Here, we study the impact of training set size on classifier performance. Specifically, we follow the suggestion to sample 5%, 50%, and 100% of the training set data for training and then observe the performance on the test set.

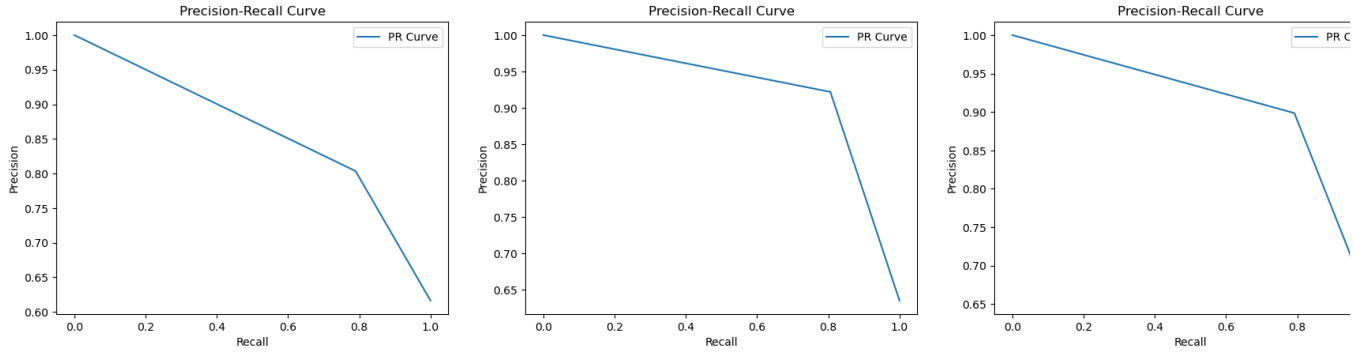
My understanding of training set size is "the total amount of data," so my approach was to take 5% and 50% of the total data, use the same process and evaluation metrics for training and testing, and then compare the final performance results on the test set.

	Accuracy	Precision	Recall	F1 Score	Kappa
5% Data	0.7319	0.7301	0.8354	0.7773	0.4373
50% Data	0.7712	0.8970	0.7330	0.8051	0.5335
100% Data	0.8129	0.8848	0.8246	0.8529	0.5965

From the first four metrics, it is evident that the model's judgment ability generally improves as the overall training data increases. Precision and recall do not strictly correlate positively with the increase in training data, but the overall trend is consistent. The Kappa statistic also steadily improves, indicating that the model's consistency in judgment also stabilizes with increasing data.



From the heatmap of the confusion matrix, it can be seen that the overall judgment ability improves with increasing data, with the top right corner, representing false positives (normal emails misclassified as spam), becoming lighter. This aligns with our desired trend.



From the PR curve, it is also evident that the area under the curve increases with the amount of data, indicating an overall improvement in recognition ability.

Overall, we can conclude that the model's performance improves in all aspects with increasing data, which aligns with our basic understanding that more information and larger training volumes lead to better judgment ability. However, a limitation in this comparison is that if we only aim to compare the impact of data volume, we should specifically tune the model parameters for each data volume to achieve the relatively optimal results under different data volumes. Due to time constraints, this was not done in this assignment.

	Training Accuracy	Training Precision	Test Accuracy	Test Precision
5% Data	0.7319	0.7301	0.8666	0.8349
50% Data	0.7712	0.8970	0.8671	0.9629
100% Data	0.8129	0.8848	0.8708	0.8865

From this comparison, we can also see that models with smaller data volumes experience more severe overfitting than models with larger data volumes, which aligns with our understanding. Another reasonable explanation is that since the parameters tuned were for the model with the largest data volume, using these parameters for models with smaller data volumes might have resulted in selecting too many feature words, further exacerbating overfitting.

III. Answer to Question Two

If the training set contains no samples satisfying (x_i = k, y = c), it means that for this feature word, its frequency of occurrence in category (c) is zero. According to the probability calculation formula of Naive Bayes, any sample containing this feature word cannot be classified into this category. However, frequency does not represent probability, and our training set cannot fully represent the whole. If we believe that this word cannot exist just because it is not in the training set, our model's ability to handle new data will be greatly reduced.

For example, if the word "holiday" never appeared in non-spam emails within our training data, our Naive Bayes model would assume that all emails containing "holiday" cannot be non-spam, implying they must be spam. However, in reality, this is unlikely to be the case, as the absence of this word in the training data does not strongly correlate it with spam.

We can solve this issue using smoothing techniques, as shown in the given formula, by adding a small value to both the numerator and denominator in the probability calculation. This way, if this problem arises again during training, we would assign it a small probability instead of zero, changing the decision from absolute to more likely, thus enhancing the robustness of our model.

IV. Answer to Question Three

In addition to the Bag-of-Words model for processing email body features, we can directly extract email header information, as detailed in the feature transformation section.

We can also use models other than Bag-of-Words to extract features, such as the more complex TF-IDF model, which considers not only the frequency of a word in the specific email but also its frequency across all emails, multiplying these two values to get the final frequency for comparison and calculation. Alternatively, we can use more complex neural network-based feature extraction methods.

We can also avoid such "automated" methods and manually select words that logically and evidently correlate well after automated classification and extraction, specifying only these for feature selection.