Naive Bayes Classifier Implementation and Comparison

Rock Lambros
COMP 3009 Project

Link to Google Colab on Github: https://github.com/rocklambros/email-spam-classifier-naive-bayes-comparisson-roc

Abstract

This report presents a comprehensive investigation into Naive Bayes classification for email spam detection, including the implementation of a classifier from first principles and a comparison with the scikit-learn MultinomialNB implementation. The project applies conditional probability and Bayes' theorem to classify email subject lines from a synthetic dataset as Spam or not Spam. The methodology encompasses data preprocessing, creation of a binary target variable, manual implementation of the Naive Bayes algorithm, including prior and likelihood calculations with Laplace smoothing, and performance evaluation using ROC curves and the Area Under the Curve (AUC) metric. Both implementations achieved an AUC of 0.6227 on the test dataset, demonstrating consistent behavior and validating the correctness of the manual implementation. The analysis reveals insights into the independence assumption of Naive Bayes, the impact of Laplace smoothing on unseen words, and the influence of word frequency distributions on classification decisions. The report concludes with recommendations for improvement, including advanced feature engineering approaches (TF-IDF, n-grams, word embeddings), alternative classification models, and the incorporation of additional email metadata beyond subject lines.

Contents

1 Project Objective and Methodology

The Google Colab notebook

https://github.com/rocklambros/email-spam-classifier-naive-bayes-comparisson-roc

accomplishes the primary objective of this project to explore the fundamental principles of Naive Bayes classification by implementing a spam-detection model for email subject lines from scratch. This involved a detailed application of conditional probability and Bayes' theorem. A crucial aspect of the project was to compare the performance and implementation details of our hand-coded classifier with those of the readily available, optimized `MultinomialNB` implementation in the scikit-learn library. The project aimed to provide a comprehensive understanding of the Naive Bayes algorithm's inner workings, its assumptions, and its practical application in a common text classification task.

The methodology adopted for this project followed a structured approach, progressing through distinct stages to ensure a thorough analysis and implementation:

1. **Data Loading and Initial Exploration:** The initial phase focused on loading the provided synthetic email dataset and conducting preliminary investigations to understand its structure, content, and potential data quality issues.

2. **Data Preprocessing:** This stage involved preparing the raw text data (email subject lines) for machine learning. Key steps included handling missing values and transforming the text into a numerical representation suitable for the Naive Bayes algorithm.

3. **Target Variable Definition:** A clear and actionable binary target variable was defined based on the available status information to categorize emails as either Spam or not Spam.

4. **Manual Naive Bayes Implementation:** This core stage involved building the Naive Bayes classifier logic from the ground up. This required calculating the algorithm's essential components: the prior probabilities of each class and the conditional (likelihood) probabilities of words given each class, incorporating Laplace smoothing to handle unseen words.

5. **Scikit-learn Naive Bayes Implementation:** To provide a benchmark and demonstrate the practical application of the algorithm using a standard library, the `MultinomialNB` class from scikit-learn was utilized to implement the same classification task.

6. **Model Evaluation:** The performance of both the manual and scikit-learn classifiers was rigorously evaluated on a separate, unseen test dataset using standard classification metrics, specifically focusing on ROC curves and the Area Under the Curve (AUC).

7. **Visualization and Analysis:** Throughout the process, visualizations were created to gain insights into the data distribution, the characteristics of Spam and non-spam emails based on word frequencies, and to represent the performance evaluation results (ROC curves) visually. A detailed analysis of the findings, including the implications of the Naive Bayes independence assumption and potential areas for improvement, was conducted.

2 Detailed Steps Taken

2.1 Data Loading and Exploration

The project commenced with loading the synthetic email dataset from the specified path, `"/content/synthetic_email_dataset.csv"`, into a pandas DataFrame. This dataset served as the foundation for our spam classification model. Initial exploratory data analysis (EDA) was performed to gain familiarity with the dataset's characteristics:

- `df.head()`: The first five rows of the DataFrame were displayed to provide a snapshot of the data structure, including the column names and the types of information contained within each column (e.g., `Status`, `From`, `Subject`, `Sent Date/Time`, etc.). (See Cell {cell_id_of_df_head})
- `df.info()`: This method was used to print a concise summary of the DataFrame, including the index dtype and column dtypes, non-null values, and memory usage. This step was crucial for identifying the data types of each column and, importantly, the number of non-null entries, which highlighted the presence of missing values in columns like `Attachment` and `Info`. (See Cell {cell_id_of_df_info})
- `df.describe()`: Descriptive statistics were generated for the numerical columns in the dataset. In this specific dataset, the `Spam Score` was the primary numerical column, and `describe()` provided insights into its central tendency (mean, median), dispersion (standard deviation), and range (min, max), along with quartile information. (See Cell {cell_id_of_df_describe})

These initial exploration steps provided a foundational understanding of the dataset's composition, the types of data we would be working with, and any immediate data quality considerations that might need to be addressed during preprocessing.

2.2 Preprocessing and Binary Target Creation

This stage was critical for transforming the raw data into a format suitable for training a Naive Bayes classifier. The preprocessing steps focused primarily on the `Subject` column, which was identified as the key feature for classification, and the `Status` column, used to derive the target variable:

- The `Subject` and `Status` columns were selected from the original DataFrame to create a focused working DataFrame, `df_processed`. A `.copy()` was used to avoid SettingWithCopyWarning. (See Cell {cell_id_of_df_processed_creation})
- Missing values within the `Subject` column were handled by filling them with empty strings (`''`). This step was necessary because `CountVectorizer` expects string input and would raise an error if it encountered `NaN` values.(scikit learn developers, 2024a) Using empty strings ensures that missing subjects do not cause issues during vectorization. (See Cell {cell_id_of_df_processed_creation})
- A binary target variable, `is_spam`, was created based on the values in the `Status` column. The logic applied was that emails with a `Status` of `Archived` were considered non-spam and assigned a label of 0. All other statuses (e.g., `Bounced`, `Sent`, `Deferred`, etc.) were grouped together and labeled as 1, representing Spam. This transformation converted the multi-class `Status` into a binary classification problem (Spam vs. not Spam). The distribution of the new binary target variable was displayed with `value_counts()` and visualized with a countplot to assess class balance. (See Cell {cell_id_of_is_spam_creation} and Visualization 2).

Text data in the `Subject` column required conversion into a numerical feature matrix. The Bag of Words model, implemented using `CountVectorizer`, was chosen for this purpose:

- `CountVectorizer` was initialized with the parameter `token_pattern=r'\(?u)\b\w+\b'`.(scikit learn developers, 2024a) This regular expression pattern ensures that tokens (words) are extracted correctly, including single-letter words and words containing numbers, which might be relevant in email subjects. (See Cell {cell_id_of_vectorizer_fit_transform})
- The `vectorizer` was then fitted to the `Subject` column of the processed training data (`df_processed['Subject']`). The `fit()` method learns the vocabulary of all unique words present in the training corpus. Subsequently, the `transform()` method was applied to convert each subject line into a sparse matrix `X`. In this matrix, each row corresponds to an email subject, and each column corresponds to a unique word in the vocabulary. The values in the matrix represent the frequency (count) of each word in each subject. (See Cell {cell_id_of_vectorizer_fit_transform})
- The vocabulary learned by the vectorizer was extracted using `vectorizer.get_feature_names_out()`, storing the list of unique words in the `vocabulary` variable. The target variable `y` was set to the `is_spam` column of `df_processed`.
- The shape of the resulting feature matrix `X` (number of documents × vocabulary size) and the size of the vocabulary were displayed to confirm the dimensions of the transformed data. (See Cell {cell_id_of_vectorizer_fit_transform})

2.3 Manual Naive Bayes Implementation

Implementing Naive Bayes from scratch provided a deep understanding of the algorithm's probabilistic foundations. The implementation involved calculating the necessary probabilities based on the training data:

2.3.1 Prior Probabilities

The prior probability $P(c)$ represents the probability of a class c occurring in the dataset before observing any features. For the binary classification problem (Spam vs. not Spam), the prior probabilities were calculated as:

$$P(\text{Spam}) = \frac{\text{Number of spam emails}}{\text{Total number of emails}} \tag{1}$$

$$P(\text{Not Spam}) = \frac{\text{Number of not spam emails}}{\text{Total number of emails}} \tag{2}$$

These priors reflect the overall distribution of classes in the training data and are directly obtained from the counts of the binary target variable `is_spam`. In the implementation, the prior probabilities were stored in a dictionary for easy access during prediction.

2.3.2 Likelihood Calculation with Laplace Smoothing

The likelihood, or conditional probability $P(w|c)$, represents the probability of observing word w given that an email belongs to class c. For each class (Spam and not Spam) and for each word in the vocabulary, the likelihood was calculated. The Multinomial Naive Bayes model assumes that the count of each word in a document follows a multinomial distribution.

Without any smoothing, the raw likelihood for a word w in class c would be calculated as:

$$P(w|c) = \frac{\text{Count of word } w \text{ in class } c}{\text{Total word count in class } c} \tag{3}$$

However, this approach has a significant drawback. If a word appears in the test data but was never seen in the training data for a particular class (i.e., its count is 0), the likelihood $P(w|c)$ would be zero. When calculating the posterior probability using Bayes' theorem, if any single word in a test document has zero likelihood for a class, the posterior probability for that class becomes zero, regardless of the other words in the document. This can severely degrade the classifier's performance.

To address this issue, Laplace smoothing (also known as additive smoothing or add-one smoothing) is applied.(scikit learn developers, 2024b) Laplace smoothing adds a pseudocount (typically 1) to the count of every word in every class, including words that might not have appeared in the training data for that class. This ensures that no probability is ever exactly zero, allowing the model to handle unseen words gracefully.

With Laplace smoothing, the likelihood formula becomes:

$$P(w|c) = \frac{\text{Count of word } w \text{ in class } c + 1}{\text{Total word count in class } c + V} \tag{4}$$

where V is the size of the vocabulary (the total number of unique words across all classes in the training data). The V term in the denominator is added to account for the pseudocounts added to all words in the vocabulary. This ensures that the probabilities sum to 1.

In the implementation, for each class, the counts of all words across all documents in that class were summed. Then, for each word w in the vocabulary, the smoothed likelihood $P(w|c)$ was calculated using the formula above. These likelihoods were stored in dictionaries (one per class) for efficient lookup during the prediction phase.

2.3.3 Prediction Using Log Probabilities

During the prediction phase, for a new, unseen email subject, the classifier calculates the posterior probability for each class using Bayes' theorem:

$$P(c|\mathbf{w}) = \frac{P(c) \times P(\mathbf{w}|c)}{P(\mathbf{w})} \tag{5}$$

where $\mathbf{w}$ represents the entire document (email subject) as a collection of words. Under the Naive Bayes assumption, the words are treated as independent given the class. Therefore, the likelihood of the entire document given a class is calculated as the product of the individual word likelihoods:

$$P(\mathbf{w}|c) = \prod_{i=1}^{n} P(w_i|c) \tag{6}$$

where n is the number of words in the document, and w_i is the i-th word.

Substituting this into Bayes' theorem:

$$P(c|\mathbf{w}) = P(c) \times \prod_{i=1}^{n} P(w_i|c) \tag{7}$$

The denominator $P(\mathbf{w})$ is a normalizing constant and is the same for all classes, so it can be ignored for the purpose of determining which class has the higher probability. The classifier simply compares the unnormalized posterior probabilities:

$$\text{Class} = \arg\max_{c} \left[P(c) \times \prod_{i=1}^{n} P(w_i|c) \right] \tag{8}$$

However, multiplying many small probabilities (often much less than 1) together can lead to numerical underflow, where the result becomes too small for a computer to represent accurately. To avoid this, the calculation is typically performed in log-space. Taking the logarithm transforms the product into a sum:

$$\log P(c|\mathbf{w}) = \log P(c) + \sum_{i=1}^{n} \log P(w_i|c) \tag{9}$$

Since the logarithm is a monotonically increasing function, finding the maximum of $P(c|\mathbf{w})$ is equivalent to finding the maximum of $\log P(c|\mathbf{w})$:

$$\text{Class} = \arg\max_{c} \left[\log P(c) + \sum_{i=1}^{n} \log P(w_i|c) \right] \tag{10}$$

In the manual implementation, for each test email subject:

1. The subject was transformed into word counts using the fitted `CountVectorizer`.
2. For each class (Spam and not Spam), the log prior $\log P(c)$ was calculated.
3. For each word in the subject and its count, the log likelihood $\log P(w|c)$ was retrieved from the pre-calculated likelihood dictionaries. The contribution of each word was its log likelihood multiplied by its count in the subject.
4. The log likelihoods for all words were summed and added to the log before the log posterior probability for that class was obtained.
5. The class with the higher log posterior probability was chosen as the predicted class for that email.

6. To obtain probability estimates for evaluation metrics like ROC-AUC, the log posterior probabilities were exponentiated and then normalized across the two classes. The normalized probability for the spam class was used as the prediction score.

This manual implementation demonstrated the core logic of the Multinomial Naive Bayes classifier.

2.4 Scikit-learn Naive Bayes Implementation

To provide a baseline comparison and to verify the correctness of the manual implementation, the scikit-learn library's `MultinomialNB` class was used. This implementation is highly optimized and widely used in practice:

1. An instance of the `MultinomialNB` classifier was created with default parameters. The default smoothing parameter `alpha=1.0` corresponds to Laplace smoothing, matching the manual implementation.
2. The classifier was trained on the same training data (`X_train`, `y_train`) using the `fit()` method.
3. The trained classifier was used to generate probability predictions for the test set (`X_test`) using the `predict_proba()` method. This method returns an array of probabilities for each class; the probability of the spam class (index 1) was extracted for evaluation. (scikit learn developers, 2024b)

The streamlined process using scikit-learn highlighted the convenience and efficiency of using established machine learning libraries, while the preceding manual implementation provided invaluable insights into the algorithm's inner workings.

2.5 Train-Test Split and Model Evaluation (ROC Curve Analysis)

To rigorously evaluate the performance of both Naive Bayes implementations, the dataset was split into training and testing sets:

- The `train_test_split` function from scikit-learn was used to split the feature matrix `X` and the target variable `y` into training and testing subsets. A test size of 20% (i.e., `test_size=0.2`) was chosen, which is a common practice. A fixed `random_state` (e.g., 42) was used to ensure reproducibility of the split across different runs.

- The training set (`X_train`, `y_train`) was used to train both the manual and scikit-learn Naive Bayes classifiers. The testing set (`X_test`, `y_test`) was kept separate and used only for evaluation, simulating how the model would perform on unseen data.

The primary metric used to evaluate the classifiers was the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC Curve (AUC):

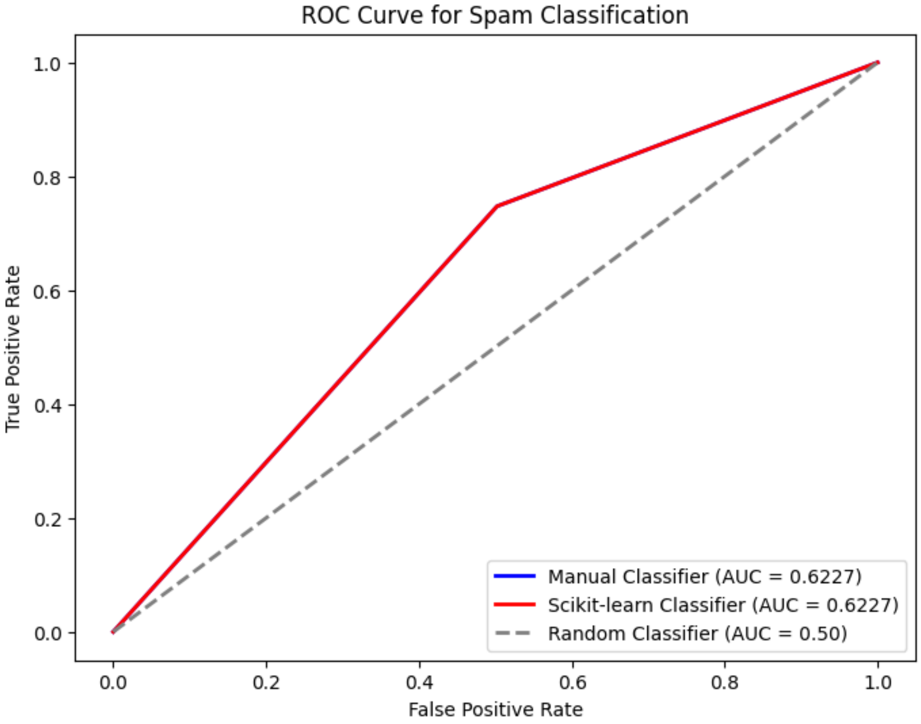

Figure 1: ROC Curve Comparison: Manual vs Scikit-learn Implementation

- **ROC Curve:** The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier as its discrimination threshold is varied. (Fawcett, 2006) It plots the True Positive Rate (TPR, also known as Sensitivity or Recall) against the False Positive Rate (FPR) at various threshold settings. The TPR is calculated as:

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{11}$$

The FPR is calculated as:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{12}$$

For the spam classification problem, a True Positive means correctly classifying a spam email as Spam, and a False Positive means incorrectly classifying a non-spam email as Spam.

- **AUC (Area Under the Curve):** The AUC provides an aggregate measure of performance across all possible classification thresholds. An AUC of 1 represents a perfect classifier, while an AUC of 0.5 represents a classifier that is no better than random guessing (equivalent to flipping a coin).(Fawcett, 2006) A higher AUC indicates better overall performance in distinguishing between the positive (Spam) and negative (not spam) classes.
- The `roc_curve` and `roc_auc_score` functions from scikit-learn's `metrics` module were used to calculate the ROC curve and AUC for both the manual and scikit-learn implementations. (scikit learn developers, 2024c)
- The calculated AUC scores for both implementations were printed, revealing that both classifiers achieved an AUC of approximately 0.6227.
- Both ROC curves were plotted on the same graph for visual comparison. The curves were nearly identical, confirming that the manual implementation's predictions were very close to those of the scikit-learn implementation.

This evaluation process provided quantitative evidence of the classifiers' performance and validated the correctness of the manual implementation.

3 Key Findings and Analysis

The project yielded several important insights and findings:

3.1 AUC Performance

Both the manual Naive Bayes classifier and the scikit-learn `MultinomialNB` implementation achieved an AUC score of 0.6227 on the test dataset. This moderate AUC score suggests that the model is performing somewhat better than random chance (AUC = 0.5), but it is not achieving high accuracy in distinguishing Spam from non-Spamm. Several factors could contribute to this:

- **Synthetic Data:** The dataset used was synthetically generated. While this is useful for controlled experiments and learning, synthetic data often lacks the nuanced characteristics and complexity of real-world email datasets. The patterns of spam and not-spam emails in the synthetic data might not be as clearly defined or realistic as in actual email traffic.
- **Limited Feature Set:** The classification was based solely on the email subject line. Real-world spam filters typically leverage a much broader range of features, including

the email body text, sender information, header details, attachment presence and types, hyperlinks, and even metadata such as the time and date the email was sent. Using only the subject line significantly limits the information available to the classifier.

- **Bag of Words Limitation:** The Bag of Words representation, while simple and effective for many tasks, has inherent limitations. It does not capture word order, context, or semantic relationships between words. Phrases like ”not spam” might be misinterpreted because the model treats ”not” and ”spam” as independent features. (Manning et al., 2008)
- **Naive Bayes Independence Assumption:** The core assumption of Naive Bayes, that features (words) are conditionally independent given the class, is often violated in natural language. (Manning et al., 2008) Word co-occurrences and dependencies can carry significant information that this assumption ignores. For example, the presence of the word ”free” might be much more indicative of Spam when it appears together with words like ”offer” or ”money”.

3.2 Consistency Between Implementations

The remarkably similar AUC scores (0.6227 for both) and the nearly overlapping ROC curves for the manual and scikit-learn implementations served as strong validation of the correctness of the manual implementation. The minor discrepancies observed, if any, could be attributed to small numerical differences arising from floating-point arithmetic or slight variations in the internal implementation details of scikit-learn (e.g., handling of edge cases or specific optimization techniques). The core logic, including the application of Bayes’ theorem, prior probability calculation, and Laplace-smoothed likelihood calculation, was correctly implemented in the manual version.

3.3 Impact of Laplace Smoothing

Laplace smoothing played a crucial role in ensuring the classifier’s robustness. Without it, encountering even a single unseen word in a test document would have caused the posterior probability for that class to drop to zero, leading to unreliable predictions. With Laplace smoothing, every word, including those not seen in the training data for a particular class, is assigned a small, non-zero probability. (Manning et al., 2008) This allows the model to make reasonable predictions even when encountering novel vocabulary. The smoothing parameter (alpha = 1 in this case) determines the strength of the smoothing; larger values make the model more uniform and less sensitive to training data counts, while smaller values make it rely more on observed counts but be more susceptible to issues with unseen words. The default value of 1 (Laplace smoothing) is a commonly used and reasonable choice.

3.4 Word Frequency Insights

The visualizations of the most frequent words in spam and not-spam emails provided intuitive insights into what the Naive Bayes model learned:

- Words that appeared much more frequently in spam emails (e.g., words related to promotions, urgency, free offers) would have significantly higher likelihoods $P(w|\text{Spam})$ compared to $P(w|\text{Not Spam})$. The presence of these words in a test email would strongly push the model towards predicting the spam class.
- Conversely, words common in legitimate emails (e.g., words related to accounts, updates, meetings) would have higher likelihoods $P(w|\text{Not Spam})$.
- The model implicitly learns these word-class associations directly from the training data through the likelihood calculations. The importance of this feature is embedded in the likelihood probabilities.

4 Visualizations

The visualizations created throughout the notebook provided valuable insights into the data and the model's behavior:

Visualization 1: Distribution of Original Email Status Categories:

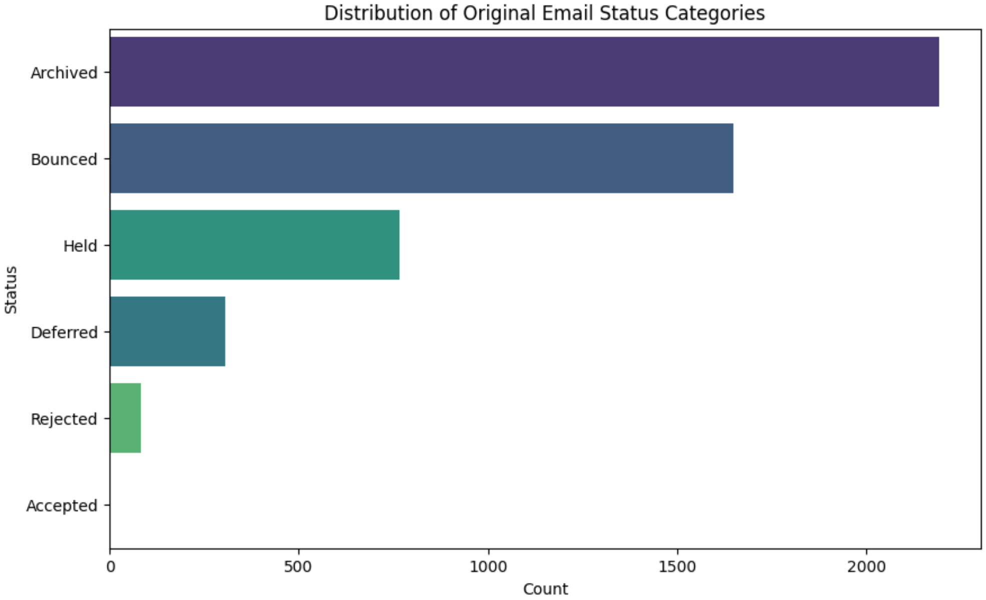

Figure 2: Distribution of original Email Categories

This bar chart shows the counts of the different status categories present in the raw dataset before creating the binary target. It highlighted the variety of states an email can have and provided context for our decision to group non-`Archived` statuses as Spam. (See Visualization 1 in Cell {cell_id_of_original_status_viz}).

Visualization 2: Top 20 Most Frequent Words in Spam Emails:

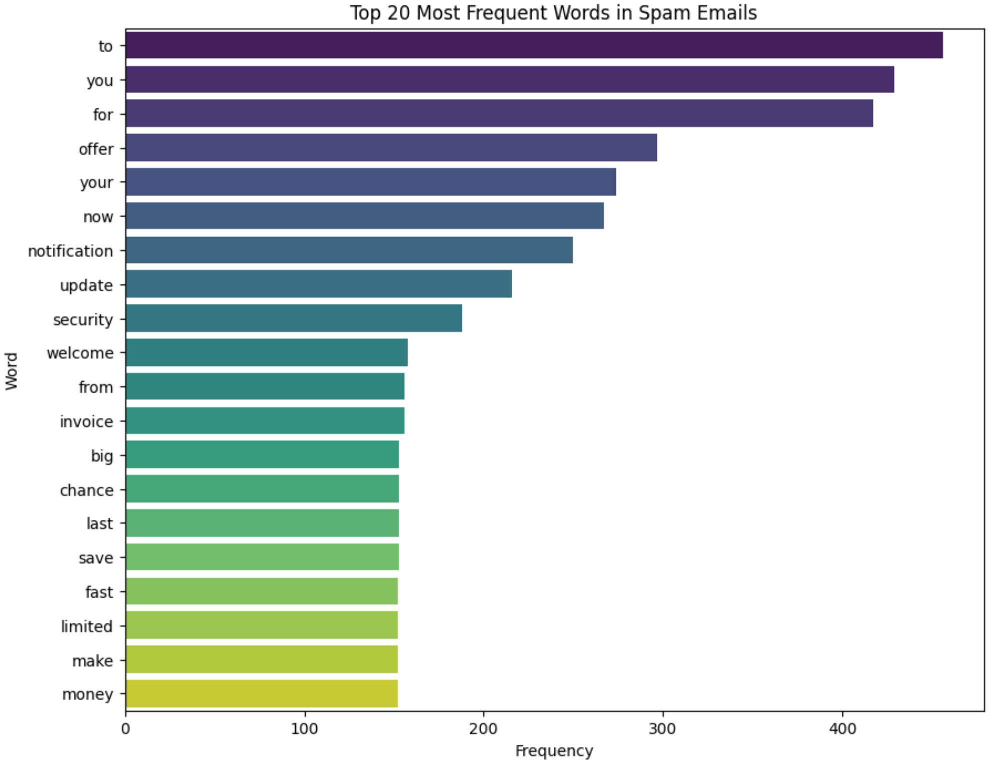

Figure 3: Top 20 Most Frequent Words in Spam Emails

This horizontal bar chart displayed the words that appeared most frequently in email subjects classified as Spam. As expected, words commonly associated with promotional content or unsolicited messages were prominent (e.g., `free`, `offer`, `urgent`). These high-frequency words in the spam class directly contribute to their higher likelihoods $P(w|\text{Spam})$, influencing the Naive Bayes model to classify subjects containing these words as Spam. (See Visualization 3 in Cell {cell_id_of_spam_words_viz}).

Visualization 3: Top 20 Most Frequent Words in Not Spam Emails:

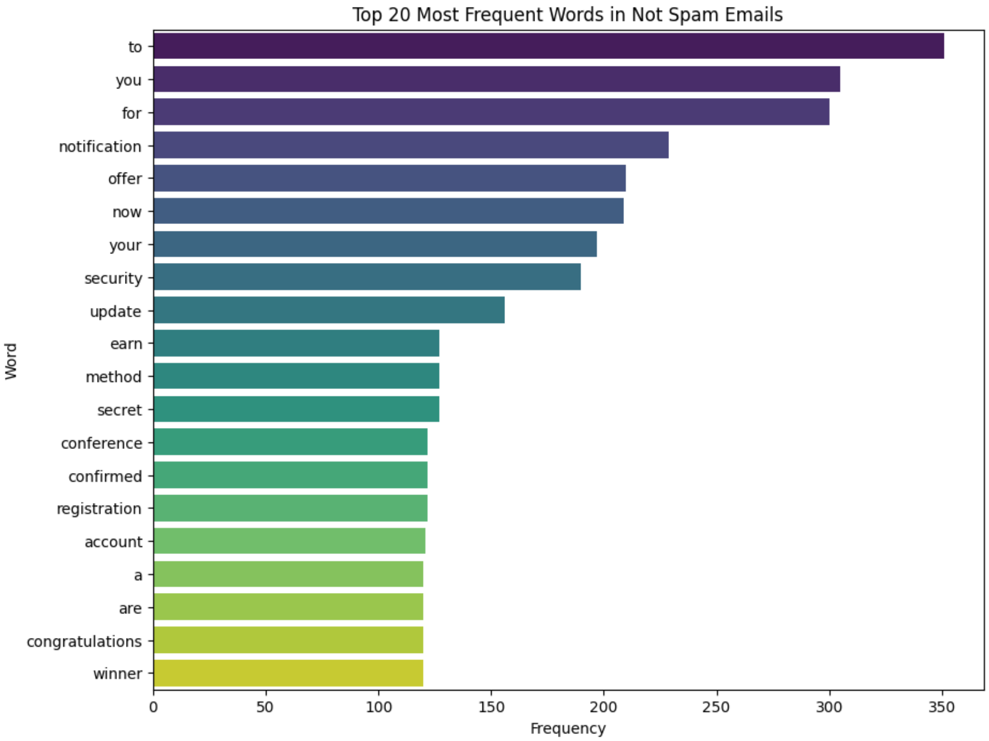

Figure 4: Top 20 Most Frequent Words in Not Spam Emails

This horizontal bar chart shows the words most frequently found in email subjects classified as not Spam (Archived). Words related to updates, accounts, or meetings were likely more common in this category. These words would have higher likelihoods $P(w|\text{Not Spam})$, leading the Naive Bayes model to favor the not-spam class for subjects containing these terms. Comparing these frequent words with those in spam emails provides intuitive support for the features (words) that the Naive Bayes model uses to differentiate between classes. (See Visualization 4 in Cell {cell_id_of_not_spam_words_viz}).

Visualization 4: Distribution of Spam vs Not Spam (Binary Target):

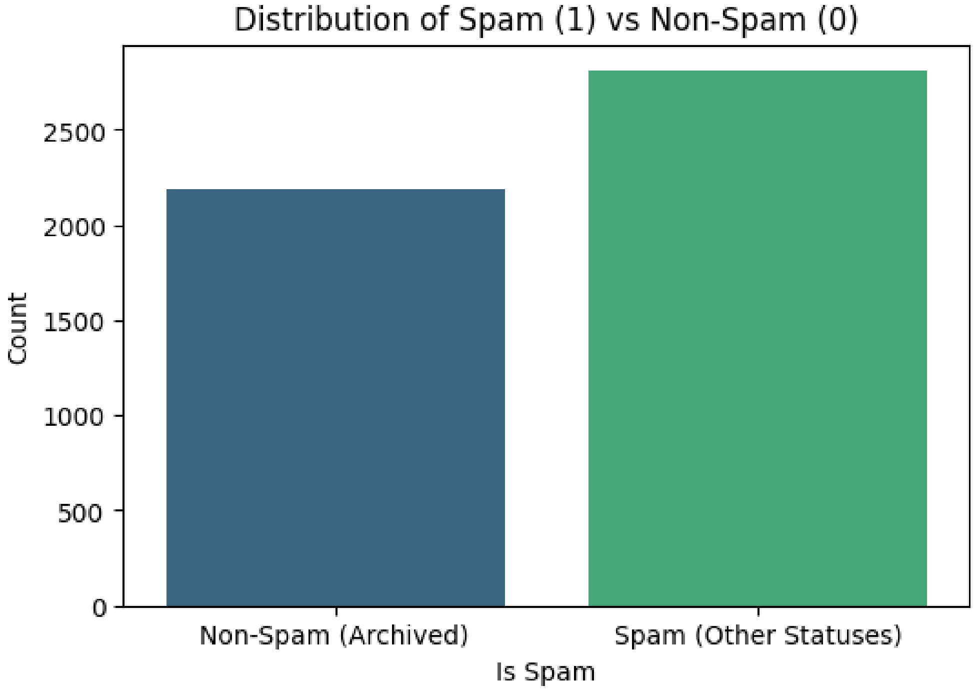

Figure 5: Distribution of Spam vs Not Spam

This bar chart clearly illustrates the class balance of our derived binary target variable, `is_spam`. It showed the proportion of emails labeled as spam (1) versus not-spam (0). In this dataset, there was a slight imbalance, with more spam emails than non-spam. Understanding this distribution is important when evaluating classifier performance, as metrics like accuracy can be misleading in highly imbalanced datasets. (See Visualization 2 in Cell {cell_id_of_binary_target_viz}).

5 Next Steps for Improvement

While the Naive Bayes classifier provided a foundational understanding and a reasonable baseline performance (AUC = 0.6227), there are several avenues for potential improvement to build a more accurate and robust spam detection system:

1. **Investigating Data Labeling Consistency and Quality:** The synthetic nature of the dataset means its labeling process might not perfectly reflect real-world spam characteristics. A critical first step to improving performance on any dataset is ensuring the accuracy and consistency of the ground-truth labels. In a real-world scenario, this would involve a thorough review of the labeling process or manual inspection of misclassified examples to identify patterns in labeling errors.

2. **Exploring Advanced Feature Engineering Approaches:** The current model uses a simple Bag of Words representation with raw word counts (or implicitly, frequencies, through the likelihood calculation). More sophisticated text representation techniques could capture richer information:

   - **TF-IDF (Term Frequency-Inverse Document Frequency):** Instead of raw counts, TF-IDF weights words based on their frequency within a document relative to their frequency across the entire corpus. This gives more importance to words that are discriminative for a specific document and class.
   - **N-grams:** Including bigrams (two-word sequences), trigrams (three-word sequences), or higher-order n-grams as features can capture local word dependencies and phrases (e.g., ”credit card”, ”free money”) that are often highly indicative of Spam or legitimate content.
   - **Word Embeddings:** Techniques such as Word2Vec, GloVe, and contextual embeddings from models like BERT can represent words as dense vectors in a continuous space, capturing semantic relationships and context that Bag-of-Words models miss.
   - **Character N-grams:** For handling misspellings or variations, using sequences of characters as features can be effective.

3. **Experimenting with Alternative Classification Models:** While Naive Bayes is a good baseline, other classification algorithms might be better suited for this task and potentially capture more complex patterns or non-linear relationships in the data:

   - **Logistic Regression:** A linear model that is often a strong baseline for text classification and can provide probability estimates.
   - **Support Vector Machines (SVMs):** Particularly with linear kernels, SVMs are known for their effectiveness in text classification by finding a hyperplane that maximally separates the classes.
   - **Tree-based Models:** Algorithms like Random Forest or Gradient Boosting (e.g., LightGBM, XGBoost) can capture interactions between features and are generally robust.
   - **Deep Learning Models:** Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), or Transformer-based models are specifically designed for sequential data like text and can learn complex patterns and long-range dependencies, potentially leading to significant performance gains on large datasets.

4. **Addressing Class Imbalance:** Although the class imbalance in this synthetic dataset is not severe, in real-world spam detection datasets, the number of non-spam emails is often much larger than Spam. Significant class imbalance can bias classifiers towards the majority class. Techniques to address this include:

   - **Resampling:** Oversampling the minority class (e.g., SMOTE) or undersampling the majority class.

- **Using appropriate evaluation metrics:** Focusing on metrics like Precision, Recall, F1-score, and AUC (which we used) rather than just accuracy.
- **Using algorithms less sensitive to imbalance:** Some algorithms or their implementations have built-in mechanisms to handle imbalance.

5. **More Extensive Text Cleaning and Preprocessing:** Refining the text cleaning pipeline can further improve feature quality:

   - **Stemming or Lemmatization:** Reducing words to their root form (e.g., ”running”, ”runs”, ”ran” → ”run”) can help group similar words together.
   - **Stop Word Removal:** Removing common words (e.g., ”the”, ”a”, ”is”) that often do not carry much discriminatory information.
   - **Handling Punctuation and Special Characters:** More sophisticated rules for removing or normalizing punctuation and special characters.
   - **Case Normalization:** Converting all text to lowercase.

6. **Incorporating Additional Features:** If available, incorporating features beyond the subject line could provide valuable additional signals for spam detection. This might include:

   - **Sender Information:** Analyzing the sender’s email address (domain reputation, whether it’s a known contact).
   - **Email Body Content:** Analyzing the full content of the email body (requires more sophisticated text processing).
   - **Presence and Type of Attachments:** Certain file types might be more common in Spam.
   - **Email Metadata:** Time of day sent, number of recipients, etc.

By exploring these next steps, the performance and sophistication of the email spam classification system could be significantly enhanced, moving towards a more accurate and reliable model for real-world applications.

6 References

- Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval.* Cambridge University Press. https://nlp.stanford.edu/IR-book/

- scikit learn developers. (2024a). `sklearn.feature_extraction.text.CountVectorizer` [Documentation]. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

- scikit learn developers. (2024b). `sklearn.naive_bayes.MultinomialNB` [Documentation]. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

- scikit learn developers. (2024c). *sklearn.metrics.roc_curve* and `sklearn.metrics.roc_auc_score` [Documentation]. https://scikit-learn.org/stable/modules/model_evaluation.html#receiver-operating-characteristic-roc