# Spamtastic! Filtering Ham-Spam Messages Using Naïve Bayes and Random Forest Algorithms

Kyla Ronquillo
Ateneo de Naga University
Naga City, Philippines
kyronquillo@gbox.adnu.edu.ph

**ACM Reference Format:**
Kyla Ronquillo. 2025. Spamtastic! Filtering Ham-Spam Messages Using Naïve Bayes and Random Forest Algorithms. In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Spam filtering is a crucial aspect of modern email communication, helping users avoid unwanted and potentially harmful messages. This paper presents an implementation of Naïve Bayes and Random Forest classifiers for effective spam detection. The study aims to compare the performance of these models using precision and recall metrics to evaluate their effectiveness in classifying messages as spam or ham.

## 2 METHODOLOGY

The spam filtering system was implemented using two machine learning algorithms: Naïve Bayes and Random Forest.

### 2.1 Naïve Bayes Classifier

The Naïve Bayes classifier is a probabilistic model based on Bayes' theorem, assuming conditional independence between features given the class label. The classification rule is given by:

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)} \tag{1}$$

where: - $P(C_k|X)$ is the posterior probability of class $C_k$ given feature set $X$, - $P(X|C_k)$ is the likelihood of the features given class $C_k$, - $P(C_k)$ is the prior probability of class $C_k$, - $P(X)$ is the probability of the feature set $X$.

To handle zero probabilities, we apply Laplace Smoothing:

$$P(x_i|C_k) = \frac{\text{count}(x_i, C_k) + \lambda}{\sum_{x \in V}(\text{count}(x, C_k) + \lambda)} \tag{2}$$

where: - $\text{count}(x_i, C_k)$ is the frequency of word $x_i$ in class $C_k$, - $V$ is the vocabulary size, - $\lambda$ is the smoothing parameter (typically set to 1 for Laplace Smoothing).

*Conference'17, July 2017, Washington, DC, USA*
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

### 2.2 Random Forest Classifier

Random Forest is an ensemble learning method that constructs multiple decision trees and aggregates their outputs to improve classification performance.

The Random Forest algorithm works as follows:

(1) Sample $m$ bootstrapped subsets from the dataset.
(2) Train a decision tree on each subset, selecting a random subset of features at each split.
(3) Aggregate the predictions from all trees using majority voting.

$$\hat{Y} = \arg\max_c \sum_{t=1}^{T} \mathbb{I}(h_t(X) = c) \tag{3}$$

where: - $h_t(X)$ is the prediction of the $t$-th tree, - $\mathbb{I}$ is an indicator function that returns 1 if the prediction matches class $c$, and 0 otherwise, - $T$ is the total number of trees.

### 2.3 Data Preprocessing

The dataset consisted of labeled messages categorized as "spam" or "ham." Preprocessing steps included:

- Removing special characters and tokenizing words.
- Keeping only alphabetic characters.
- Creating a vocabulary of unique words.
- Counting occurrences of each word in ham and spam messages.

**Listing 1: Data Preprocessing Function**

```
def preprocess_data(text):
    # Keep alphabet and space only
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Lowercase all of them
    text = text.lower()
    # Split into words
    words = text.split()
    # Remove stopwords and apply stemming
    words = [stemmer.stem(word) for word in
        words if word not in stop_words]
    return words
```

### 2.4 Naïve Bayes Classifier

The Naïve Bayes classifier was implemented to classify messages as either spam or ham. The key steps involved:

- Applied Laplace Smoothing ($\lambda = 1$) to handle zero probabilities and improve generalization.
- Computed prior probabilities for spam and ham messages.
- Counted word frequencies in spam and ham messages using a dictionary structure.
- Classified test messages based on log probabilities to prevent numerical underflow.

**Listing 2: Counting Word Frequencies in Spam and Ham Messages**

```
# Separate messages by class
ham_messages = df_sms[df_sms['label'] == '
    ham']['tokens']
spam_messages = df_sms[df_sms['label'] == '
    spam']['tokens']

# Initialize word count dictionaries
ham_word_counts = defaultdict(int)
spam_word_counts = defaultdict(int)

# Count word occurrences in ham messages
for tokens in ham_messages:
    for word in tokens:
        ham_word_counts[word] += 1

# Count word occurrences in spam messages
for tokens in spam_messages:
    for word in tokens:
        spam_word_counts[word] += 1
```

The above code snippet processes a dataset of labeled messages by separating them into two categories: spam and ham. It initializes two dictionaries to store word frequencies and iterates through each category, counting occurrences of each word. This step is crucial for later probability estimation in Naïve Bayes.

**Listing 3: Computing Word Probabilities for Spam and Ham Classification with Laplace Smoothing**

```
# Vocabulary size (number of unique words)
V_size = len(V)

# Total word count in each class
total_ham_words = sum(ham_word_counts.values
    ())
total_spam_words = sum(spam_word_counts.
    values())

# Compute word probabilities with Laplace
    Smoothing
ham_probabilities = {word: (ham_word_counts[
    word] + 1) / (total_ham_words + V_size)
    for word in V}
spam_probabilities = {word: (
    spam_word_counts[word] + 1) / (
    total_spam_words + V_size) for word in V
    }
```

The second snippet calculates the probability of each word occurring in spam and ham messages using Laplace Smoothing. This technique prevents zero probabilities by adding 1 to each word count. The computed probabilities are essential for making predictions using the Naïve Bayes classifier.

## 3 RESULTS AND DISCUSSION

The classifiers were evaluated using precision and recall:

**Table 1: Performance Metrics of Classifiers**

| Model | Precision | Recall |
|---|---|---|
| Naïve Bayes | 0.83 | 0.94 |
| Random Forest | 1.00 | 0.79 |

**Key Observations:**

- **Naïve Bayes:** Achieved a high recall (94%), meaning it effectively identified most relevant instances but had a lower precision (83%), indicating a higher number of false positives.
- **Random Forest:** Exhibited perfect precision (100%), meaning all positive predictions were correct, but had a lower recall (79%), indicating that some relevant instances were missed.

**Implications of Precision and Recall:** The trade-off between precision and recall has significant implications depending on the application. A model with high recall, such as Naïve Bayes, is beneficial when it is crucial to identify as many relevant instances as possible, even at the cost of false positives. This is suitable for applications like medical diagnosis, where missing a disease (false negative) is more harmful than misidentifying a healthy patient (false positive).

On the other hand, a model with high precision, such as Random Forest, is preferred when false positives must be minimized. This is useful in applications like fraud detection, where falsely labeling legitimate transactions as fraudulent can lead to unnecessary inconveniences for users. However, its lower recall means some fraudulent cases may go undetected.

**Effects of Misclassification:** Misclassifications can lead to different consequences depending on whether they are false positives or false negatives. In spam detection, for instance, a false positive (ham classified as spam) could cause important messages to be lost, while a false negative (spam classified as ham) could lead to users receiving unwanted emails. Choosing the appropriate balance between precision and recall depends on the specific requirements of the application.

## 4 CONCLUSION

Both classifiers have strengths and weaknesses. Naïve Bayes is lightweight and effective, while Random Forest offers higher accuracy. Future work could explore hybrid models that combine the advantages of both approaches for enhanced spam detection and explore techniques for optimizing the balance between precision and recall, such as threshold tuning or hybrid models, to improve classification performance in specific use cases.