
Filtering Spam E-Mails

A Gaussian Naive Bayes Approach

Huynh Minh Khang - SE192197¹

¹FPT University

March 26, 2025

ABSTRACT

This report presents a comprehensive study on the implementation of a spam filtering system using the Gaussian Naive Bayes algorithm. The objective is to accurately classify emails as spam or ham by leveraging probabilistic methods underpinned by statistical theory. The document details each stage of the process—from data ingestion and preprocessing through feature extraction and model evaluation. In addition, the report elucidates the theoretical foundations of Bayes' Theorem and its application within a Gaussian framework. Experimental results demonstrate that the proposed approach attains competitive accuracy with efficient training and inference times.

Keywords. Email · Spam · Naive Bayes' · Gaussian Naive Bayes'

1 Introduction

Spam emails continue to challenge both individual users and organizations, as they inundate inboxes and pose potential security risks. The present study explores the application of the Gaussian Naive Bayes algorithm—a probabilistic classifier noted for its simplicity and computational efficiency—to the task of spam detection. Despite perceptions of Naive Bayes as a simplistic method, when implemented correctly, it demonstrates robust performance in real-world settings.

This report delineates the methodology, from data collection and preprocessing to model training and evaluation, and discusses the underlying probabilistic theory. Emphasis is placed on the systematic removal of noise from email text, as well as the mathematical justification for using Gaussian assumptions in continuous feature spaces.

2 Mathematical Background

Bayes' Theorem is a cornerstone of probability theory and provides a mechanism for updating prior beliefs in light of new evidence. The theorem is expressed as:

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}$$

In the context of spam filtering:

- **Posterior Probability ($P(A | B)$):** The probability that an email is spam given the observed features.

- **Likelihood ($P(B | A)$):** The probability of observing the given features assuming the email is spam.
- **Prior Probability ($P(A)$):** The initial probability of an email being spam.

Naive Bayes algorithms calculate these probabilities under the assumption of feature independence, selecting the class with the highest posterior probability. In the Gaussian variant, each continuous feature is assumed to follow a normal distribution, thereby accommodating numerical data effectively.

For continuous features, the likelihood for each feature x_i given class C is modeled as:

$$P(x_i | C) = \frac{1}{\sqrt{2\pi\sigma_{C,i}^2}} \exp\left(-\frac{(x_i - \mu_{C,i})^2}{2\sigma_{C,i}^2}\right)$$

Assuming feature independence, the joint likelihood for the feature vector $X = \{x_1, x_2, \dots, x_n\}$ is:

$$P(X | C) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma_{C,i}^2}} \exp\left(-\frac{(x_i - \mu_{C,i})^2}{2\sigma_{C,i}^2}\right)$$

For numerical stability, the logarithm of the likelihood is taken:

$$\log P(X | C) = -\frac{1}{2} \sum_{i=1}^n \left(\log(2\pi\sigma_{C,i}^2) + \frac{(x_i - \mu_{C,i})^2}{\sigma_{C,i}^2} \right)$$

The decision rule for classification is thus:

$$\begin{aligned} \hat{C} &= \operatorname{argmax}_C \left[\frac{P(C)P(X | C)}{P(X)} \right] \\ &= \operatorname{argmax}_C [P(C)P(X | C)] \\ &= \operatorname{argmax}_C [\log(P(C)P(X | C))] \\ &= \operatorname{argmax}_C [\log P(C) + \log P(X | C)] \end{aligned}$$

3 Data Acquisition and Consolidation

The study utilizes three datasets, each containing emails labeled as spam (1) or ham (0). The datasets are sourced from multiple repositories to ensure diversity. The following procedure describes data ingestion, cleaning, and consolidation:

```

1  import pandas as pd
2
3  data1 = pd.read_csv("data/lingSpam.csv")
4  data2 = pd.read_csv("data/enronSpamSubset.csv")
5  data3 = pd.read_csv("data/completeSpamAssassin.csv")
6
7  data1.drop("Unnamed: 0", inplace=True, axis=1)
8  data2.drop(["Unnamed: 0", "Unnamed: 0.1"], inplace=True, axis=1)
9  data3.drop("Unnamed: 0", inplace=True, axis=1)
10 data = pd.concat([data1, data2, data3], axis=0)

```

```
11
12 data.dropna(inplace=True)
```

The resulting dataset comprises approximately 18,650 entries, each with a body text and a corresponding label.

Table 1: Example data

Id	Body	Label
0	Subject: great part-time or summer job ! ...	1
1	Subject: auto insurance rates too high ? ...	1
2	Subject: do want the best and economical hunti...	1
3	Subject: email 57 million people for \$ 99 ...	1
4	Subject: don't miss these ! attention ! ...	1


4 Text Preprocessing and Feature Engineering

Textual data undergoes extensive preprocessing to enhance the performance of the classification model. The steps are as follows:

4.1 Link Removal

Hyperlinks are removed from the email text using regular expressions to eliminate irrelevant tokens.


```
1 import re
2
3 x = data["Body"]
4 x_clnd_link = [re.sub(r"http\S+", "", text) for text in x]
```

 Python

4.2 Removal of Non-Alphanumeric Characters

Unwanted characters are stripped from the text using the following pattern:


```
1 pattern = "[^a-zA-Z0-9]"
2 x_cleaned = [re.sub(pattern, " ", text) for text in x_clnd_link]
```

 Python

4.3 Lowercasing

Text is converted to lower case to ensure uniformity:

```
1 x_lowered = [text.lower() for text in x_cleaned]
```

 Python


4.4 Tokenization

The text is segmented into tokens using NLTK's word tokenization function:

```

1 import nltk
2
3 x_tokenized = [nltk.word_tokenize(text) for text in x_lowered]

```

 Python


4.5 Lemmatization

Tokens are lemmatized using NLTK's WordNet Lemmatizer, which reduces words to their canonical forms:

```

1 nltk.download("wordnet")
2 from nltk.stem import WordNetLemmatizer
3
4 lemma = WordNetLemmatizer()
5 x_lemmatized = [[lemma.lemmatize(word) for word in text] for text in x_tokenized]

```

 Python


4.6 Stopword Removal

Stopwords are removed to retain only the most informative tokens:

```

1 stopwords = nltk.corpus.stopwords.words("english")
2 x_prepared = [[word for word in text if word not in stopwords] for text in
  x_lemmatized]

```

 Python

The number of unique tokens is then computed to evaluate the vocabulary size.


4.7 Bag-of-Words Feature Extraction

A bag-of-words representation is constructed using scikit-learn's CountVectorizer, limiting the feature set to 20,000 terms:

```

1 from sklearn.feature_extraction.text import CountVectorizer
2
3 vectorizer = CountVectorizer(max_features=20000)
4 x_features = vectorizer.fit_transform([" ".join(text) for text in
  x_prepared]).toarray( )

```


 Python

Subsequently, the dataset is partitioned into training and testing subsets:

```

1 from sklearn.model_selection import train_test_split
2 import numpy as np
3
4 x_train, x_test, y_train, y_test = train_test_split(
5     x_features, np.asarray(data["Label"]), random_state=42, test_size=0.2
6 )

```

 Python

5 Model Development: Gaussian Naive Bayes

The Gaussian Naive Bayes classifier is employed to model the probability distributions of continuous features. The model training and evaluation are executed as follows:

5.1 Model Training

During training, the Gaussian Naive Bayes classifier estimates the necessary parameters from the training data for each class C_k and each feature i .

(i) **Class Prior:** The prior probability of class C_k is calculated as:

$$P(C_k) = \frac{N_{C_k}}{N}$$

where N_{C_k} is the number of training samples in class C_k and N is the total number of training samples.

(ii) **Mean ($\mu_{C_k,i}$):** The mean of feature i for class C_k is computed by:

$$\mu_{C_k,i} = \frac{1}{N_{C_k}} \sum_{j:y_j=C_k} x_{j,i}$$


where x_{i_j} is the i^{th} feature value of the j^{th} sample in class C_k .

(iii) **Variance ($\sigma_{C_k,i}^2$):** The variance of feature i for class C_k is given by:

$$\sigma_{C_k,i}^2 = \frac{1}{N_{C_k}} \sum_{j:y_j=C_k} (x_{j,i} - \mu_{C_k,i})^2$$

Below is the corresponding Python snippet that performs model training:

```
1 import time
2 from sklearn.naive_bayes import GaussianNB
3
4 start_time = time.time()
5 NB = GaussianNB()
6 NB.fit(x_train, y_train)
7 end_time = time.time()
8 print(round(end_time - start_time, 2))
```

 Python

The training process is computationally efficient, completing in approximately 3.51 seconds.

5.2 Model Evaluation

First, the confusion matrix is computed and visualized as a heatmap to provide insight into the distribution of prediction errors. Let:


- TP denote the number of true positives
- TN denote the number of true negatives
- FP denote the number of false positives
- FN denote the number of false negatives

The confusion matrix is defined as:

$$\begin{pmatrix} \text{TN} & \text{TP} \\ \text{FN} & \text{TP} \end{pmatrix}$$

The following code computes and visualizes the confusion matrix:

```
1 from sklearn.metrics import confusion_matrix
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
```

 Python

```

5  y_pred = NB.predict(x_test)
6
7  cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
8
9  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
10 plt.xlabel("Predicted")
11 plt.ylabel("Actual")
12 plt.show()

```

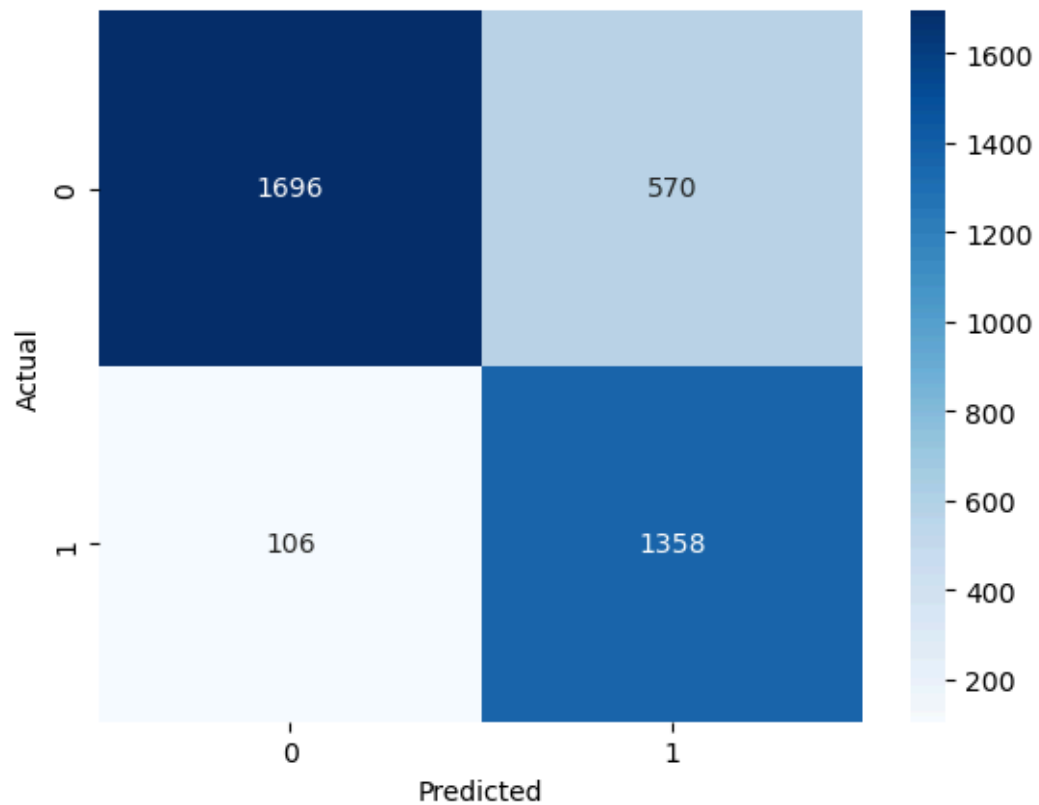


Figure 1: Confusion Matrix

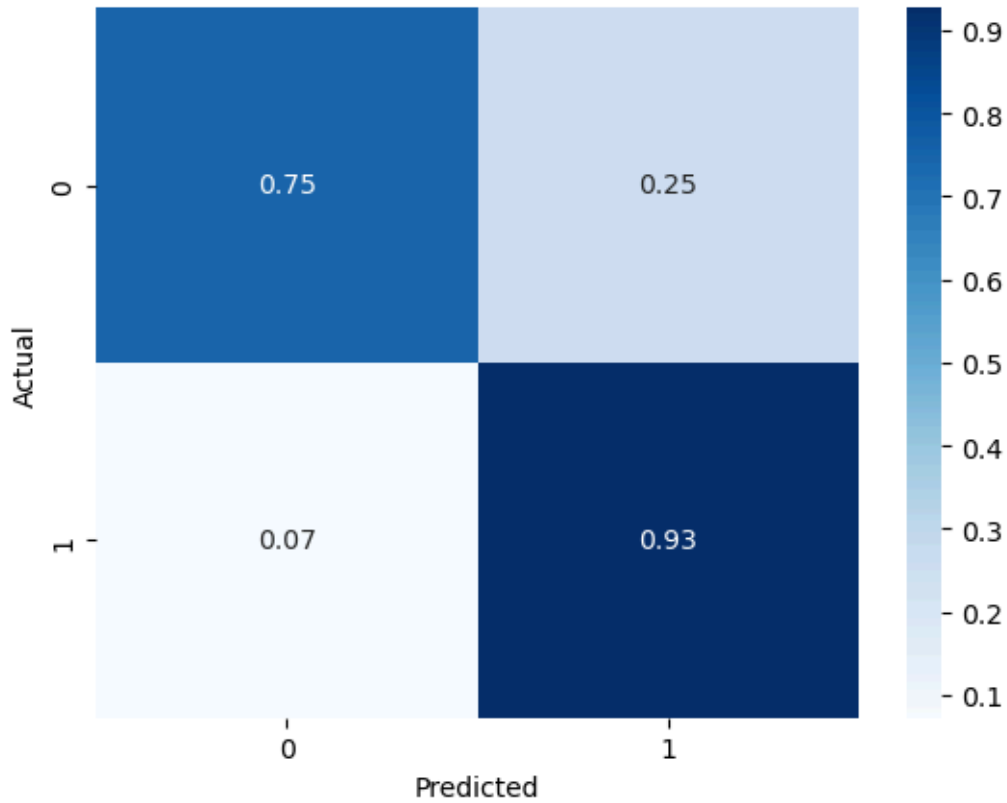


Figure 2: Normalized Confusion Matrix

After visualizing the confusion matrix, we calculate key evaluation metrics. The evaluation metrics are defined as follows:

- Accuracy:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- F1-Score:

$$\text{F1-Score} = 2 \cdot \frac{1}{\text{Precision}^{-1} + \text{Recall}^{-1}}$$

where:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The corresponding code computes these metrics:

```
1 from sklearn.metrics import f1_score
2
3 print("Accuracy:", NB.score(x_test, y_test))
4 print("F1-score:", f1_score(y_test, y_pred))
```

Python

Based on the run, the model achieved an accuracy of approximately 82% and an F1-Score of approximately 80%.

6 Conclusion and Future Work

This study provided a systematic exploration of a spam filtering system built upon the Gaussian Naive Bayes algorithm. The report detailed the stages of data ingestion, rigorous text preprocessing, and feature extraction via a bag-of-words model. The Gaussian Naive Bayes classifier, underpinned by sound probabilistic theory, was trained and evaluated, yielding an accuracy of approximately 82% and f1-score of 80%.

Future research directions may include:

- Integrating additional features such as email metadata.
- Comparing alternative classifiers (e.g., support vector machines, ensemble methods).
- Investigating deep learning techniques for improved performance on larger datasets.

References

- [1] “scikit-learn Documentation..” [Online]. Available: <https://scikit-learn.org/>
- [2] “Natural Language Toolkit (NLTK).” [Online]. Available: <https://www.nltk.org/>
- [3] Metsis, Vangelis and Androutsopoulos, Ion and Paliouras, and Georgios, *Spam Filtering with Naive Bayes - Which Naive Bayes?*. 2006.