# Naive Bayes Spam Filter

Bren Jay Magtalas

## I. Introduction

THE Naive Bayes is a classification algorithm based on Bayes' theorem. It provides a way of discriminating between classes by calculating the probability of a hypothesis given the observed data. The prior belief is updated with the new evidence presented. The algorithm is called naive due to the assumption that each data feature is independent of each other. In the context of a spam filter, the algorithm becomes a binary classification problem with the data labeled as spam or ham. The Naive Bayes formula is given by

$$P(\omega \mid x_1, x_2, \ldots, x_d) = \frac{\prod_{i=1}^{d} P(x_i \mid \omega) P(\omega)}{\sum_{\omega} \prod_{i=1}^{d} P(x_i \mid \omega) P(\omega)} \tag{1}$$

where $d$ is the vocabulary size $|V|$, $\omega$ is the class label spam or ham (i.e., $\omega \in ham, spam$), and $x_i$ is the presence or absence of words in the dataset.

It is important to take note that the specific implementation of the algorithm used is the Bernoulli Naive Bayes, where all the features can only take on the value of 1 if the word is in the document, and 0 if it is not. The interest is in $P(\omega \mid x_1, x_2, \ldots, x_d)$ or the probability that an email is spam or ham given that the words $x_i$ are present or not. The $P(\omega)$ is called the class prior probability. It is the proportion of a specific class $\omega$ with the total number of labeled data. The class conditional probability or $P(x_i \mid \omega)$ is the probability of the word $x_i$ appearing given that the email is of class $\omega$. For the spam filter, it can be calculated by

$$P(x_i \mid \omega) = \frac{\sum_{D \in D_\omega} I(x_i \in D)}{|D_\omega|}$$

where $D$ is the set of all emails or documents, $D_\omega$ is the set of documents belonging to class $\omega$, $|D_\omega|$ is the cardinality or the total number of documents of class $\omega$, and $I(x_i \in D)$ is the indicator function taking on values of 1 or 0 depending on whether the word $x_i$ is in the document $D$.

The conditional probability can be interpreted as the proportion of the number of documents of class $\omega$, where the specific word $x_i$ occurs, and the total number of documents in that class. Because of the naive assumption that all the words are independent of each other, the conditional likelihood can be obtained by just multiplying all the probabilities for all $d$.

When a word does not occur in a document class of the training data, the numerator becomes zero, which poses a problem because the whole likelihood also becomes zero, leading to incorrect predictions. The absence of the word in a document does not mean that it will never appear in any document of that class. To mitigate this problem Lambda Smoothing is utilized, with the modified formula for class conditional probability given by

$$P(x_i \mid \omega) = \frac{[\sum_{D \in D_\omega} I(x_i \in D)] + \lambda}{|D_\omega| + \lambda |V|} \tag{2}$$

The training phase of the algorithm can be considered as the calculation of the prior probabilities and conditional probabilities for all words in the vocabulary in the training set for spam and ham classes. Equation (1) is then used to classify the testing set into spam or ham. Because the main concern is to get the value of $\omega$ that maximizes the probability of the document belonging to this class, considering that the denominator of equation (1) is the same for all classes, the only contributing factor in getting maximum probability when comparing all classes is the numerator. The decision rule based on the maximum a posteriori (MAP) is given by

$$\omega_{MAP} = \arg\max_{\omega} P(\omega) \prod_{i=1}^{d} P(x_i \mid \omega) \tag{3}$$

The multiplication of probabilities can oftentimes cause problems due to loss in precision. Because the quantities are less than 1, the likelihood function grows smaller as the number of features increases. For large datasets and in general, it would be better to get the log-likelihood of the equation (3). Getting the natural logarithm of the function would still yield the same result because the logarithm is a strictly increasing function. Zero probabilities are also taken care of in the lambda smoothing making the function defined in every possible probability (the logarithm of zero is undefined). When taking the logarithm, instead of multiplying probabilities, the equation becomes the sum of logarithms given by

$$\omega_{MAP} = \arg\max_{\omega} \log(P(\omega)) + \sum_{i=1}^{d} \log(P(x_i \mid \omega)) \tag{4}$$

## II. Implementation

### A. Data Pre-processing

The dataset is a subset of the TREC06 Public Spam Corpus often used to benchmark spam algorithms. The folder consists of a *labels* file that contains the path to the messages and the corresponding class labels. There is also a folder named *data* with a subdirectory containing subdirectories of email messages. Each email message has relevant information such as metadata, email header, email body, and attachments which can be viewed in text form.

In order to obtain the class labels and file path, the *labels* file is accessed and separated by *' ../'* because the main program is located in the same folder as the dataset. The labels are stored as *'spam'* or *'ham'* and the data path is in the form *'data/xxx/xxx'*.

The initial data in lists of strings of words are obtained by iterating through all the path directories. The file corresponding to the path is read as text and split using regular expression. To keep it simple, the delimiters used are just white spaces, commas, periods, and newlines. By utilizing the python `re` package, only words containing the sequence of alphabetic characters [a-zA-Z] are considered. Furthermore, the words are lowercased and because the interest is only the existence of words, only the unique words per email are used as the data. After splitting and sifting through the data, some entries are just empty lists. Because these data may not provide meaningful information to the model and can potentially just introduce noise, the empty data and its corresponding labels are removed from the dataset.

When the cleaning is done, the data is then split into training and test set. The dataset is partitioned into 70-30 randomly by getting first a random permutation of indices corresponding to the size of the examples. Obtaining the randomized index will ensure that the training and test document set will have its corresponding label.

### *B. Training*

The training function for the Naive Bayes implementation accepts three variables as input, the training data, the training label, and an optional list of predefined vocabulary of words, which will be utilized when the algorithm is improved later on. The training data and labels are converted into a numpy array for easier data manipulation. If there is no vocabulary given, the words that will be used in training will come from the training data. Each training example was iterated through and the unique words in all of the given data were sorted and stored in an array that represents the $V$ mentioned earlier.

The prior probabilities $P(\omega)$ were obtained by first getting the number of spam and ham documents using boolean indexing, summing over each class, and dividing by the total number of documents. This would give the proportion of spam and ham emails in the training examples. To get the count of each vocabulary word in spam and ham documents, the process was simplified by converting all words in the training example into a one-dimensional array of spam and ham documents. This was done to avoid looping through each document and individually checking the existence of words, adding to the complexity and memory usage of the algorithm. The occurrence of each word is then counted resulting in arrays of spam word count and ham word count with both at the same size as the vocabulary.

Because different values of lambda are to be tested, to avoid repeating the training steps, the likelihood computation with lambda smoothing is moved over to the prediction function. The output of the training function is the prior likelihoods, word count, document count, and vocabulary.

### *C. Testing*

The prediction function for the algorithm accepts the testing data, testing labels, training outputs, and the value of lambda. In the early testing of the function, it was discovered that the most time-consuming part of the function is identifying whether each word in the vocabulary is in the testing data. Instead of accepting the raw testing data as a list of examples with a list of words, it is more efficient to preprocess the data into a list of boolean arrays. In each testing example, an array with the same size as the vocabulary is populated with True for words that are present in the example, and False for those that are absent.

The conditional probabilities with lambda smoothing are then calculated element-wise through the vectorized operations. Because the conditional probabilities are only calculated when the word appears in a class, the probability of when it does not appear is calculated by subtracting the conditional probabilities of each word in each class from 1. The natural logarithm of each conditional probability is then calculated.

For each boolean array representing the data in the testing set, the log-likelihood corresponding to the probability of each word appearing or not, determined by the boolean array, is summed up together with the prior probabilities. The one with the maximum calculated probability is identified to be the predicted class of the algorithm. The output of the prediction function is the arrays of actual labels and predicted labels.

### *D. Evaluation Metrics*

The evaluation metrics chosen for the classification algorithm are Accuracy, Precision, and Recall. Accuracy determines the proportion of emails that Naive Bayes can correctly classify. Precision gives a measure of safety and it is the proportion of emails classified as spam that are actually spam. Recall gives a measure of effectiveness and it is the proportion of actual spam that is correctly classified. The formulas for the evaluation metrics are given by

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

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

where

*TP* (true pos) : number of spam messages classified as spam
*TN* (true neg) : number of ham messages classifies as ham
*FP* (false pos) : number of ham messages classified as spam
*FN* (false neg) : number of spam messages classified as ham

In testing the model, 5 different values of $\lambda$ are used ($\lambda = 2.0, 1.0, 0.5, 0.1, 0.005$). The value with the best precision and recall are printed but the $\lambda$ used for the rest of the implementation is the one with the highest precision. Precision is preferred because fewer misclassified ham messages are much more desirable.

### *E. Classifier Modifications*

The former implementation of the Naive Bayes algorithm takes all the words in the training set as its vocabulary and each word is used in the classification process. There are words however that can be considered less informative than others

and would provide less discriminating factors in the algorithm. To find the 200 most informative words, techniques suggested by Hovold are utilized [1].

There are two methodologies tested. The first one is the use of Mutual Information and the second one is the Mutual Information in conjunction with the reduction of frequent and infrequent words. The Mutual Information function accepts all the data, labels, number of desired informative words, and whether the reduction of words is used or not. It starts off with the same training function to get the necessary values to calculate the mutual information given by

$$\sum_{x}\sum_{\omega} P(x,\omega)\log\frac{P(x,\omega)}{P(x)P(\omega)} \tag{5}$$

where $P(x,\omega)$ is the joint probability of the word $x$ and class $\omega$. Because of conditional independence, it is analogous to the conditional probability in equation (2), and the same lambda smoothing is applied. The $P(\omega)$ is already calculated in the training as a prior class probability. The of the word occurring or not $P(x \in \{1,0\})$ can be obtained by adding the word count in spam and ham, dividing it by the vocabulary size for class 1, and subtracting the result from one for class 0.

By utilizing the property of logarithms, equation (5) can be further simplified as

$$\sum_{x}\sum_{\omega} P(x,\omega)(\log P(x,\omega) - \log P(x) - \log P(\omega)) \tag{6}$$

The mutual information of each word gives a measure of how relevant the word is in discriminating between spam and ham classes. The top 200 values corresponding to the most informative words are obtained and saved as the new vocabulary. This vocabulary would be the additional argument that will be included in the training function.

Another method implemented in conjunction with mutual information is the reduction of words by removing frequent and infrequent words. In the same mutual information function additional argument *reduce_words* is turned True. Using the combined word count obtained earlier, words occurring less than 3 times or words occurring more than 200 times are identified. To incorporate this in the mutual information calculation, the mutual information value of elements satisfying the removal criteria is set to 0. In this way, the frequent and infrequent words are ranked last for the most informative words.

## III. Results and Discussion

The training of the Naive Bayes algorithm, without specifying the vocabulary of words, took approximately 3 minutes on the local computer used. The total vocabulary size is found to be 82977 words. The prior probability for spam is 0.6581 and 0.3419 for ham. The number of emails containing spam is greater in number than those that are ham.

Converting the testing set into a boolean array took 15 minutes. It was the slowest out of all the functions executed. This task was supposed to be included in the testing function but because of the number of lambda that needs to be tested, separating it into a different function proved to be a huge time saver.

| Metrics | Accuracy | Precision | Recall |
|---|---|---|---|
| lambda=2.0 | 0.9900 | 0.9912 | 0.9936 |
| lambda=1.0 | 0.9907 | 0.9938 | 0.9921 |
| lambda=0.5 | 0.9893 | 0.9945 | 0.9893 |
| lambda=0.1 | 0.9805 | 0.9896 | 0.9808 |
| lambda=0.005 | 0.9694 | 0.9704 | 0.9837 |

TABLE I
PERFORMANCE METRICS WITH LAMBDA SMOOTHING

| Metrics | Mutual Info | MI+reduced |
|---|---|---|
| Accuracy | 0.8593 | 0.8817 |
| Precision | 0.9128 | 0.8561 |
| Recall | 0.8698 | 0.9866 |

TABLE II
PERFORMANCE METRICS WITH LAMBDA SMOOTHING

The testing of the model involves trying out 5 different values of lambda smoothing. The classification run-time only takes seconds for every lambda. The summary of the values obtained can be found in TABLE 1. The lambda with the maximum accuracy is 1.0, maximum precision is 0.5, and maximum recall is 2.0. Ideally, both precision and recall should be as high as possible, but in the case of the spam filter, it would be better to classify some spam messages as ham than classify ham messages as spam. Because higher precision can be traded off with slightly lower recall, the value of lambda that maximizes the precision is chosen to be used for the rest of the testing. The values for all the performance metrics are very high indicating an excellent classification performance across all lambda.

When applying modifications to the algorithm the execution time of the training and testing functions is significantly reduced, all of it takes just seconds. However, the time it takes to calculate and obtain the top 200 most informative words takes roughly 5 minutes, which is still significantly faster than the prior implementation. The summary results for the two modification methodology is summarized in TABLE 2.

Using Mutual Information alone yields high precision, good accuracy, and good recall. If mutual information is used combined with word reduction, the model yields a very high recall, good accuracy, and good precision. Mutual information model with reduced words has also higher accuracy. Although it was stated that precision is more important than recall, when examining the top informative words in the models, MI includes single letters, prepositions, conjunctions, and pronouns, which at a glance seemed to be less informative words. The words in MI+reduced however include words that are more obvious when identifying spam such as premium, unclaimed, and winner.

## IV. Conclusion

The Bernoulli Naive Bayes algorithm produced a very good classification performance when all the vocabulary words in the training set is used. The lambda equal to 0.5 is taken as the smoothing value due to the highest precision it yielded. Because of the large number of features that need to be considered the execution time of the algorithm is slower. To improve performance, the top 200 words were obtained and used as the new vocabulary through mutual information and

a combination of mutual information and word reduction. Using MI alone provided higher precision but seemingly less informative words. The MI+reduced provided higher accuracy, recall, and much more obvious spam words. The reason for the higher recall in MI+reduced might be the higher number of spam words in the vocabulary due to the higher proportion of spam documents.

For future works, it is highly recommended to perform cross-validations to make sure that the partitioning of data is not biased. Getting an equal proportion of spam and ham documents may also improve performance when using a smaller vocabulary. Better data cleaning methods such as removing HTML tags, excluding email headers, and parsing non-utf-8 encoded data are matters that can be looked into.

## REFERENCES

[1] Johan Hovold, Naive Bayes Spam Filtering using Word Position Attributes. In *Conference on Email and Anti-Spam*, Stanford University, 2005.