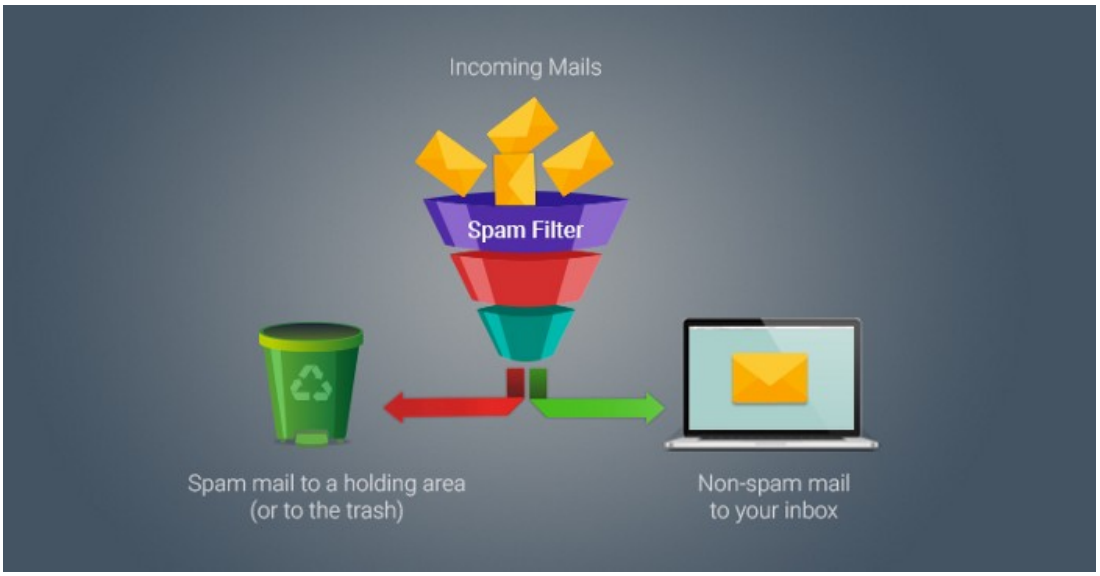
PRML ASSIGNMENT - 03

SPAM CLASSIFIER

Sushane Parthan CS20M003

Priyanka Bedekar CS20M050



1 INTRODUCTION

Probability is one of the major branches in mathematics. It is often used to solve many real-life situations. Mathematicians spent years building practical models to fit in the growing issues. Within this area, conditional probability represents the probability of an event happening in relation to the occurrence of another event. It assumes that no prior prediction can be made on the probability that an event is going to happen. This fundamental concept of conditional probability lies on the basis of Bayes' Theorem. Bayes' Theorem was first discovered by Thomas Bayes in 1763.

Bayes' Theorem gives us the ability to compute the unknown conditional probability of one pair of events given the known independent probability of each event and the reverse conditional probability of this pair of events.

WHAT IS NAIVE BAYES' THEOREM: It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes' classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

In the field of Spam Classification, Naive Bayes' classifiers work by correlating the use of tokens (typically words, or other independent features), with known spam and non-spam e-mails and then using **Bayes' theorem** to calculate a probability that an email is or is not spam.

BAYES' THEOREM: In probability theory and statistics, Bayes' theorem (alternatively Bayes' law or Bayes' rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

Bayes' theorem is stated mathematically as the following equation:

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

Where A and B are events and $P(B) \neq 0$.

- $P(A \mid B)$ is a conditional probability: The likelihood of event A occurring given that B is true.
- $P(B \mid A)$ is also a conditional probability: The likelihood of event B occurring given that A is true.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B respectively; they are known as the marginal probabilities.
- A and B must be different events.

2 DATASET: EXPLORING THE DATASET

The Ling-Spam dataset is a collection of 2893 spam and non-spam messages curated from the Linguist List, which is a well known online resource in the field of linguistics. These messages focus on linguistic interests around job postings, research opportunities and software discussion. The ham messages were obtained by randomly downloading digests from the list's archives, while the spam messages were received by an individual.

The Ling-Spam dataset consists of 2412 non-spam and 481 spam emails. From this extended dataset, we have created a training dataset of 962 emails of which 481 are Spam mails (i.e 50%) and 481 are Non-Spam (i.e 50%). The reason for considering an equal number of spam and ham emails is to ensure that our Classifier is unbiased, and does not assume any prior distribution on Spam/ Non-Spam mails.

We have divided the entire data set into Training and Test Datasets as follows:

1. **Train Set:** Total: 962, Spam: 481 (50.0%), Non-Spam: 481 (50.0%)
2. **Test Set:** Total: 2893, Spam: 481 (16.626%), Non-Spam: 2412 (83.374%)

In our dataset, all 481 spam mails are uniquely named, and can be distinctly recognized as each spam filename begins with the same five characters *spmsg*.

3 PREPROCESSING DATA

In any text mining problem, text cleaning is the first step where we remove those words from the document which may not contribute to the information we want to extract. Emails may contain a lot of undesirable characters like punctuation marks, stop words, digits, etc which may not be helpful in detecting the spam email.

The emails in our dataset have been preprocessed in the following ways:

- **Removal of stop words** – Stop words like "and", "the", "of", etc are very common in all English sentences and are not very meaningful in deciding spam or legitimate status, so these words have been removed from the emails.
- **Lemmatisation** – It is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. For example, "include", "includes," and "included" would all be represented as "include". The context of the sentence is also preserved in lemmatisation.

With respect to our algorithm, dataset preprocessing involves removal of non-words, numbers, words containing digits mixed with characters, and special characters from the mail files. There are several methods of implementation of such preprocessing, from which we explore the use of CountVectorizer, as is explained below.

4 CREATING THE DICTIONARY

The format of our dataset is such that the first line of the mail is subject and the third line contains the body of the email. We will only perform text analysis on the contents of the mail to identify features for detection of spam mails. As a first step, we need to create a dictionary of words by splitting individual words and adding them to our feature space, whilst taking care not to repeat identical words (same features).

Once each individual word is split, we use **CountVectorizer** from Scikit-Learn module to identify unique features. We then create our dictionary as a list of unique feature names identified by iterating through all words in all training mails. While using CountVectorizer, we have ensured that the stop words "all", "in", "the", "is", and "and" are not included in features.

Our Dictionary contains a total of **21885** words, which is the size of our feature space.

```
def createDict():
```

This is the function in our code which creates the dictionary. It takes each .txt file from our Training Dataset (Train_Data) which is the email and splits it to obtain separate words. The function then removes multiple occurrences of words (features), and removes single length as well as numeric/alphanumeric words.

5 FEATURE EXTRACTION

Feature extraction involves extracting separate word count vectors (which is our feature count vector) of 21885 dimensions for each email from the training set. Each individual word count vector contains the frequency of 21885 dictionary words pertaining to that specific file in the training set. Most of them will be zero, for reasons explained below.

Consider an example where we have 1000 words in our calculated dictionary. Each word count vector contains the frequency of 1000 dictionary words in the particular training file. Let the email in example contain the text **"Complete PRML today, complete it today!"**. Then its word count vector will be encoded as:

$$[0,0,0,0,0,\ldots,0,0,2,0,0,0,\ldots,0,0,1,0,0,\ldots,0,0,1,0,0,\ldots,2,0,0,0,0,0]$$

Here, the word counts of the training mail we have considered are placed at (suppose) 296^{th}, 359^{th}, 615^{th}, and 995^{th} index of word count vector of length 1000, while the other indices remain 0.

```
def wordsinAllMails(dict1):
```

This is the function which creates the word count vector for all training mails. This function in turn invokes:

```
def wordsinOneMail(dict1, inputMail):
```

Where $dict1$ is a list containing our dictionary and $inputmail$ is the content of a single email text file.

These functions count the occurrence of all dictionary words in each email, and store it as a vector in the matrix $words\_matrix$. This is a matrix of size $n \times d$ (962 x 21885), where n is the total number of training mails and d is the dimension of our feature space (equal to number of words in our dictionary).

We can reduce the size of our dictionary by using the "top" 10000 words (10000 most commonly used words) out of the total of 21885 words. It will reduce the size of our $words\_matrix$. This trade-off is between computational efficiency and accuracy. As we reduce the number of dictionary words, computational efficiency will increase leading to faster computation and hence classification, whilst reducing accuracy of our spam classifier since we have fewer features to match.

6 Training the Classifier

Naive Bayes is a conditional probability model: Given a problem instance to be classified, represented by a vector $w = (w_1, w_2, \ldots, w_d)$ representing some d features (independent variables), it assigns to this instance probabilities $P(C_k|(w_1, w_2, \ldots, w_d))$, of k possible classes.

The problem with the above formulation is that if the number of features d is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as $P(C_k|w) = \frac{P(C_k)P(w|C_k)}{P(w)}$.

6.1 Naive Bayes algorithm

Let the data (emails) be $\{x_1, x_2, \ldots, x_n\} \in \{0,1\}^d$ where d is the number of words in the dictionary. Let the True Labels be $\{y_1, y_2, \ldots, y_n\} \in \{0,1\}$.
Here, n = Total number of mails which is 962 and d = Total number of features (dictionary words) which is 21885.

The list of vectors $words\_matrix$ of size $n \times d$ stores the count of the occurrence of each dictionary word in an email, as a vector represented as rows of this Matrix.

```
def trainNB(words_matrix):
```

The above function takes as parameters $words\_matrix$ (Count of occurrence matrix). We use the function to calculate our $2d+1$ parameters, namely Probability of Spam mails ($pSpam$), Probability of Spam words P_j^1 (pSpamWords), and Probability of Ham words P_j^0

(pHamWords) where $j \in d$ and $Labels \in \{0,1\}$.

Calculating pSpam:

```
def trainLabel():
```

The above function identifies training emails which are spam by checking filenames which start with *spm*, and returns their true labels (y_i), stored in $email\_label$ list. Values in $email\_label[i]$ are such that $y_i = 1$ for Spam and $y_i = 0$ for Non-Spam emails.

We set the probability of Spam emails (**pSpam**) by calculating the fraction of Spam mails out of all mails. Total number of spam emails will be equal to sum of entries in $email\_label$ vector.

$$P(Spam) = \frac{\text{Total no of Spam mails}}{\text{Total no of mails}}$$
$$P(Spam) = \frac{\sum_i^n (y_i = 1)}{n}$$
$$P(Non-Spam) = 1 - P(Spam)$$

Calculating pSpamWords and pHamWords:
pSpamWords, **pHamWords**: These parameters are two d- dimensional vectors, which store the probabilities of each dictionary word in Spam and Non-Spam mails.

To avoid the possibility of zero probabilities for dictionary words, we ensure that each dictionary word occurs at least once. That is, we set the $d \times 1$ vectors $dict\_ham\_count$ and $dict\_spam\_count$ as arrays of $1s$ initially to account for at least single occurrences of all words in ham and spam mails respectively.

This technique is called Laplace Smoothing, where we set all occurrences to 1.

$$P(Word_j|Spam) = \frac{P(Word_j) * P(Spam|Word_j)}{P(Spam)}$$
$$P(Word_j|Non-Spam) = \frac{P(Word_j) * P(Non-Spam|Word_j)}{P(Non-Spam)}$$

7 Prediction: Classification of Test Mails

Let P_1 be $P(Y_{test} = 1|X_{test})$ and P_2 be $P(Y_{test} = 0|X_{test})$.

$$\text{If } P(Y_{test} = 1|X_{test}) > P(Y_{test} = 0|X_{test}) = \begin{cases} 1, & \text{Spam} \\ 0, & \text{Non-Spam} \end{cases}$$

Where:

$$\begin{aligned} P(Y_{test}=1|X_{test}) &\propto P(X_{test}|Y_{test}=1) * P(Y_{test}=1) \\ &\propto \left(\Pi_{j=1}^{d} P(\text{Fraction of times the } j^{th} \text{ word occurs in Spam mails})\right) * P(Spam) \\ &\propto \left(\Pi_{j=1}^{d} (P\_SpamWords[j]^{\text{Total count of occurrence of x\_j}})\right) * P(Spam) \\ &\propto \left(\Pi_{j=1}^{d} (P(Word_j|Spam))^{\text{Total count of occurrence of x\_j}}\right) * P(Spam) \end{aligned}$$

Taking logarithm:

$$= \left(\sum_{j=1}^{d} (\text{Total count of occurrence of} x_j) * \ln P(Word_j|Spam)\right) + \ln P(Spam)$$

Similarly:

$$\begin{aligned} P(Y_{test}=0|X_{test}) &\propto P(X_{test}|Y_{test}=0) * P(Y_{test}=0) \\ &= \left(\sum_{j=1}^{d} (\text{Total count of occurrence of} x_j) * \ln P(Word_j|Non-Spam)\right) + \ln P(Non-Spam) \end{aligned}$$

8 Calculating the Accuracy

We compare and study the predicted values with actual values of our known test set to measure how accurate our Spam-Classifier is with respect to classifying emails.
The test dataset we have used to check our accuracy metric is attached in the Zip file as T

To make the measurement, we'll use accuracy defined as the following metric:

$$\boxed{\textbf{Accuracy} = \frac{\text{Number of correctly classified mails}}{\text{Total number of classified mails}}}$$

We calculated our classifier accuracy as a percentage of total number of classified test mails as follows:

$$\begin{aligned} \textbf{Accuracy (\%)} &= 100 - \left(\frac{\text{Number of misclassified emails}}{\text{Total number of classified mails}} * 100\right) \\ &= 100 - \left(\frac{\text{Sum of absolute values of difference between legitimate and classified labels}}{\text{Total number of classified mails}} * 100\right) \end{aligned}$$

Number of correctly classified mails = 2879
Total number of classified mails = 2893

$$\begin{aligned} \textbf{Accuracy} &= \frac{2879}{2893} * 100 \\ &= 99.516\% \end{aligned}$$

Just for Comparison, The inbuilt BernoulliNB function gives the accuracy:

```
[Running] python -u "c:\Users\Admin\Desktop\PRML-3\Library.py"
[[2400   12]
 [   6  475]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2412
           1       0.98      0.99      0.98       481

    accuracy                           0.99      2893
   macro avg       0.99      0.99      0.99      2893
weighted avg       0.99      0.99      0.99      2893
```

9 Reference

1. All acknowledgements for the dataset go to the original authors of "A Memory-Based Approach to Anti-Spam Filtering for Mailing Lists". The dataset was made publicly available as a part of that paper.
2. Wikipedia: Naive_Bayes_Classifier.