Scientific Computing, Modeling & Simulation
Savitribai Phule Pune University

Master of Technology (M.Tech.)
Programme in Modeling and Simulation

Internship Project Report

Spam Email Filter using Naive Bayes Algorithm

Prashik Taware
MT2132

Academic Year 2022-23

Scientific Computing, Modeling & Simulation
Savitribai Phule Pune University

Certificate

This is certify that this report titled

Spam Email Filter using Naive Bayes Algorithm

authored by

Prashik Taware (MT2132)

describes the project work carried out by the author under our supervision during the period from September 2022 to November 2022. This work represents the project component of the Master of Technology (M.Tech.) Programme in Modeling and Simulation at the Department of Scientific Computing, Modeling & Simulation, Savitribai Phule Pune University.

Mihir Arjunwadkar, Faculty
SCMS-SPPU, Pune, India

Scientific Computing, Modeling & Simulation
Savitribai Phule Pune University

Author's Declaration

This document titled

Spam Email Filter using Naive Bayes Algorithm

authored by me is an authentic report of the project work carried out by me as part of the Master of Technology (M.Tech.) Programme in Modeling and Simulation at the Department of Scientific Computing, Modeling & Simulation, Savitribai Phule Pune University. In writing this report, I have taken reasonable and adequate care to ensure that material borrowed from sources such as books, research papers, internet, etc., is acknowledged as per accepted academic norms and practices in this regard. I have read and understood the University's policy on plagiarism (`http://unipune.ac.in/administration_files/pdf/Plagiarism_Policy_University_14-5-12.pdf`).

Prashik Taware
MT2132

Abstract

E-mail spam continues to become a problem on the Internet. Spammed e-mail may contain many copies of the same message, commercial advertisement or other irrelevant posts like pornographic content. In previous research, different filtering techniques are used to detect these e-mails such as using Random Forest, Naive Bayesian, Support Vector Machine (SVM) and Neutral Network. In this research, we test Naive Bayes algorithm for e-mail spam filtering on two datasets and test its performance, i.e., Spam Data and SPAMBASE datasets. The performance of the datasets is evaluated based on their accuracy, recall, precision and F-measure.The result shows that the type of email and the number of instances of the dataset has an influence towards the performance of Naive Bayes

Acknowledgements

I am thankful to **Prof. Mihir Arjunwadkar** for his guidance during this project. I would like to thank him for his support in analysis and algorithm design. I would also like to thank everyone who shared greta reference articles with me and empowered me to do this kind of work independently.

Contents

Chapter 1

Introduction

Nowadays, email provides many ways to send millions of advertisements at no cost to the sender. As a result, many unsolicited bulk emails, also known as spam emails, spread widely and become a serious threat to not only the Internet but also to society. For example, when a user receives a large amount of email spam, the chance of the user forgetting to read a non-spam message increases. As a result, many email readers have to spend their time removing unwanted messages. Email spam also may cost money to users with dial-up connections, waste bandwidth, and may expose minors to unsuitable content. Over the past many years, many approaches have been provided to block email spam [1]. For filtering, some email spam are not being labeled as spam because the email filtering does not detect that email as spam. Some existing problems are regarding accuracy for email spam filtering that might introduce some error. Several machine learning algorithms have been used in spam email filtering, but Na¨ıve Bayes algorithm is particularly popular in commercial and open-source spam filters [2]. This is because of its simplicity, which makes them easy to implement and just need short training time or fast evaluation to filter email spam. The filter requires training that can be provided by a previous set of spam and non-spam messages. It keeps track of each word that occurs only in spam, in non-spam messages, and in both. Naive Bayes can be used in different datasets where each of them has different features and attributes.

The objectives of this project are as follows
(i) to implement the Naive Bayes algorithm for email spam filtering on a particular dataset having spam emails and non spam emails.
(ii) to evaluate the performance of Naive Bayes algorithm for email spam filtering on the chosen datasets.

1.1 Methodology

This section describes the methodology that is used for this project. The methodology for this project mainly revolves around the concept of Naive Bayes , checking the overall occurrence of a specific word over all the datasets i.e to check how spammy a given word is and the last is to perform smoothing, which is a technique which will be described in the later part of the report.

1.1.1 Naive Bayes Classifier

The Naive Bayes algorithm is a simple probabilistic classifier that calculates a set of probabilities by counting the frequency and combination of values in a given dataset [4]. In this project, Naive Bayes classifier uses a bag of words features to identify spam email and a text is represented as the bag of its word. The bag of words is always used in methods of document

classification, where the frequency of occurrence of each word is used as a feature for training classifiers. This bag of words features are included in the chosen datasets. Naive Bayes technique used Bayes theorem to determine the probabilities of spam email. Some words have particular probabilities of occurring in spam email or non-spam email. Example, suppose that we know exactly, that the word Free could never occur in a non-spam email. Then, when we saw a message containing this word, we could tell for sure that it was a spam email. Bayesian spam filters have learned a very high spam probability for the words such as Free and Viagra, but a very low spam probability for words seen in non-spam email, such as the names of friend and family member. So, to calculate the probability that email is spam or non-spam Naive Bayes technique used Bayes theorem as shown in the formula below.

Where:

- $P(spam\,|word)$ is probability that an email has a particular word given the email is spam.
- P(spam) is the probability that any given message is spam.
- $P(word\,|spam)$ is the probability that the particular word appears in a spam message.
- P(non-spam) is the probability that any particular word is not spam.
- $P(word\,|non-spam)$ is the probability that the particular word appears in non-spam.

We will be referring to non-spam emails as ham emails.

$$P(Spam\,|\,word) = \frac{P(word\,|\,Spam)\,P(Spam)}{P(word)}$$

$$P(Non-spam\,|\,word) = \frac{P(word\,|\,Non-spam)\,P(Non-spam)}{P(word)}$$

1.1.2 Spammy Word

We have a way to compute the probability of an email being spam using relatively simple terms, but it's not yet clear how to compute those conditional probabilities. This classifier works by taking a large number of emails that have already been hand-labeled as spam or ham, and using that data to compute word spam probabilities, by counting the frequency of each word. Imagine we're given a training set of 5000 randomly chosen emails. We examine them and label 2000 of them as spam emails and 3000 as ham emails (so P(S) = 0.4, and P(H) = 0.6). We'd like to calculate $P(Vagra\,|S)$ and $P(Viagra\,|H)$. The easiest thing to do would be to count how many spam emails have "viagra" and divide that by the total number of spam emails, and do the same thing for ham. So, if there were 216 spam emails with "viagra" and 12 ham emails with "viagra", we'd say

$P(Viagra\,|S) = 217/2002 = 11$

$P(Viagra\,|H) = 13/3002 = 0.43$

1.1.3 Smoothing

There's a small problem while building a Spam Filter using Naive Bayes Classifier : what if there's a word (say, "Pokemon") that we've only ever seen before in ham emails, and not spam? In that case, $P(Pokemon\,|S) = 0$, and the entire spam probability will go to zero, because we're multiplying all of the word probabilities together and we've never seen "Pokemon" in spam mail before. A malicious spammer, knowing this weakness, could just slip the word "Pokemon" in the pill-pushing email, and it would get right past our classifier. We would like to be robust to words we haven't seen before, or at least words we've only seen in one setting. The solution is

to never let any word probabilities be zero, by smoothing them upwards. Instead of starting each word count at 0, start it at 1. This way none of the counts will ever have a numerator of 0. This overestimates the word probability, so we also add 2 to the denominator. (We add 2 because we're implicitly keeping track of 2 things: the number of emails that contain that word, and the number that don't. The sum of those two things should be in the denominator,

The smoothed word probabilities for the previous example are now

$P(Viagra\ |S) = 217/2002 = 11$

$P(Viagra\ |H) = 13/3002 = 0.43$

The estimate for "Pokemon" in spam emails would now be 1 / 2002 instead of 0.

This technique is called Laplace smoothing and was used in the 18th century to estimate the probability that the sun will rise tomorrow.

1.1.4 Overall Flow of the Project

- The main aim of this project is to build a Spam Filter which will classify the emails as spam emails and ham emails. The Spam Filter will function upon the Naive Bayes Algorithm
- To set up our classifier , we will be required to train our classifier over a set of previously recorded data or a dataset.
- The dataset on which our classifier will be trained upon will be having both spam and ham emails. In order to feed our classifier a relatable and uncomplicated data we will be required to pre-process our dataset. Preprocessing will include cleaning and visualizing the data at hand.
- The next step will be to select our features , this will be done by creating a vocabulary list which contains the most frequent 2500 words which appear throughout all the emails.
- Feature selection will be carried out by building a sparse matrix which will give us the count of the occurrence of each of the words in the vocabulary list. This information will be furthermore useful for training our classifier.
- The dataset at our hand will be further split into training and testing datasets and then we will test the performance of our Spam Filter over the test data and will evaluate its performance with the help . The following figure can help visualize the overall flow of the project

1.2 Algorithm

1. Load the dataset
2. Create a function to read all the emails from a dataset and preprocess each email accordingly.
3. Create a Dataframe for all the emails in the dataset and create a vocabulary list for 2500 words
4. Split the dataset into training set and test data set
5. Create grouped Sparse matrix for the words in the vocabulary list for the emails in training dataset and record the occurences
6. Create a series for category training set

7. Perform 5th and 6th step for test data as well
8. Create a full matrix from grouped sparse matrix and category list from training set and calculate probabilities based on the recorded occurrences for words in vocabulary predict and classify the test data as based on the calculated probabilities
9. visualise a decision boundary and results
10. Calculate performance metrics

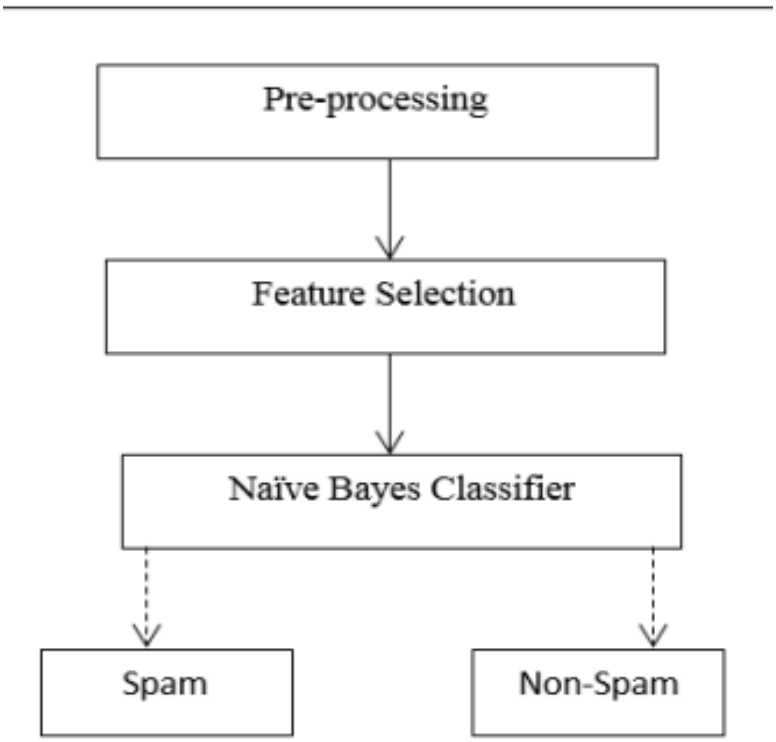


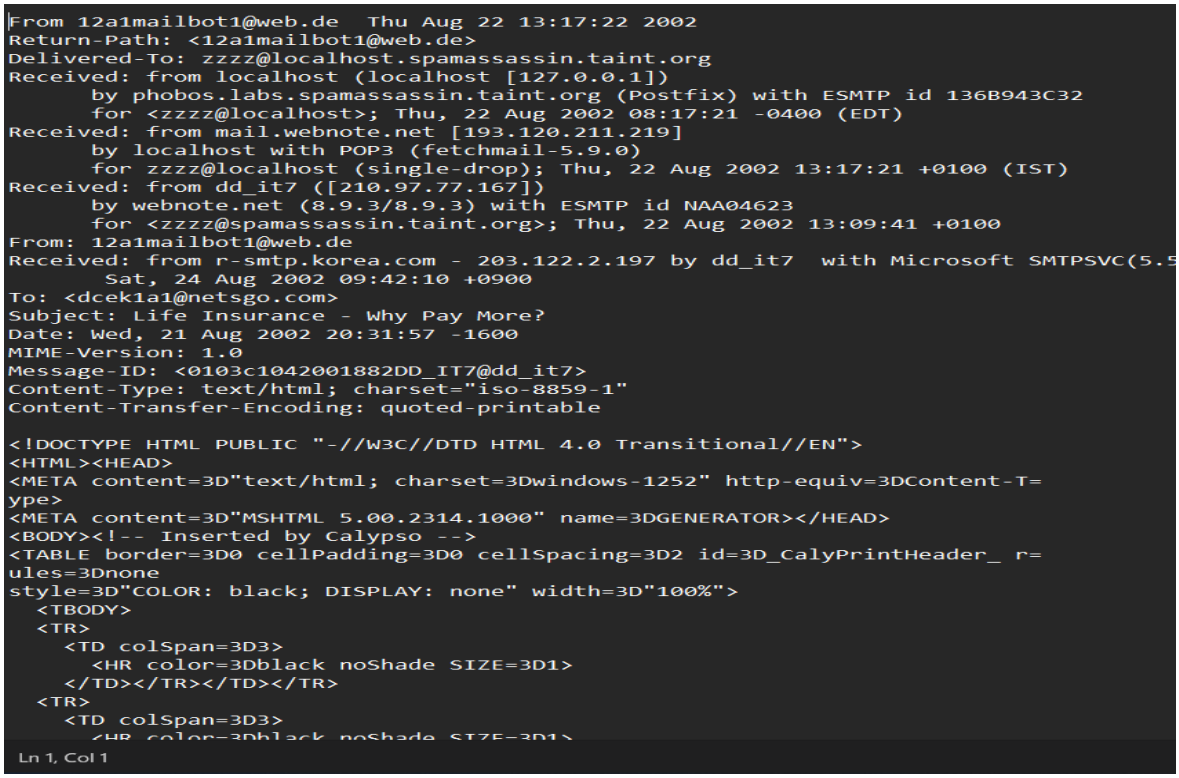
Figure 1.1: Overflow of the Project

1.3 Dataset Information

1.3.1 Summary of the dataset

The Dataset that we will be working on comes from an open source Anti Spam Platform called Spam-Assassin. The website has a huge dataset of spam emails and ham emails which are made available for the public to build Spam Filter similar to what is being built in this project. Once the dataset has been downloaded , we will get a corpus of spam emails and emails.ham

The dataset we are working upon has a collection of 1896 Spam emails and 3900 ham emails. Overall we are working with a sum total of 5796 emails. Each having their own email bodies with different contents. To view a particular email file from this dataset we can use any text editor . Below is a sample email which was opened using notepad ."



```
From 12a1mailbot1@web.de  Thu Aug 22 13:17:22 2002
Return-Path: <12a1mailbot1@web.de>
Delivered-To: zzzz@localhost.spamassassin.taint.org
Received: from localhost (localhost [127.0.0.1])
      by phobos.labs.spamassassin.taint.org (Postfix) with ESMTP id 136B943C32
      for <zzzz@localhost>; Thu, 22 Aug 2002 08:17:21 -0400 (EDT)
Received: from mail.webnote.net [193.120.211.219]
      by localhost with POP3 (fetchmail-5.9.0)
      for zzzz@localhost (single-drop); Thu, 22 Aug 2002 13:17:21 +0100 (IST)
Received: from dd_it7 ([210.97.77.167])
      by webnote.net (8.9.3/8.9.3) with ESMTP id NAA04623
      for <zzzz@spamassassin.taint.org>; Thu, 22 Aug 2002 13:09:41 +0100
From: 12a1mailbot1@web.de
Received: from r-smtp.korea.com - 203.122.2.197 by dd_it7  with Microsoft SMTPSVC(5.5
       Sat, 24 Aug 2002 09:42:10 +0900
To: <dcek1a1@netsgo.com>
Subject: Life Insurance - Why Pay More?
Date: Wed, 21 Aug 2002 20:31:57 -1600
MIME-Version: 1.0
Message-ID: <0103c1042001882DD_IT7@dd_it7>
Content-Type: text/html; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<META content=3D"text/html; charset=3Dwindows-1252" http-equiv=3DContent-T=
ype>
<META content=3D"MSHTML 5.00.2314.1000" name=3DGENERATOR></HEAD>
<BODY><!-- Inserted by Calypso -->
<TABLE border=3D0 cellPadding=3D0 cellSpacing=3D2 id=3D_CalyPrintHeader_ r=
ules=3Dnone
style=3D"COLOR: black; DISPLAY: none" width=3D"100%">
  <TBODY>
  <TR>
    <TD colSpan=3D3>
      <HR color=3Dblack noShade SIZE=3D1>
    </TD></TR></TD></TR>
  <TR>
    <TD colSpan=3D3>
      <HR color=3Dblack noShade SIZE=3D1>
```

Ln 1, Col 1

Figure 1.2: Preview of a sample email from Dataset

Chapter 2

Pre-Processing

As mentioned before , for this project we will be following the path as shown in Fig.1. The data we are working with is still raw in nature , meaning , in order to get the most out of it , a lot of pre-processing is required to do on the dataset.

2.1 Data Visualization of spam and non spam emails

2.1.1 Accessing the data

The dataset after downloading will give us a corpus of both spam emails and ham emails. These emails will be stored into four folders namely spam1 ,spam2, ham1, ham2. In order to access these data files we will have to supply the absolute path of these files and store it into a variable . The two main tasks that we have to be aware of while proceeding further are :

- We are mainly concerned about the content of each email i.e the main body of the email and not the overall structure like the header and footer or any other content.
- We need to access each of these emails and build a dataframe of the overall data we have .

We can tackle the first task by defining a generator function which will iterate over all the 5796 emails we have in our dataset and it will extract the Email body from each one of it. The second task can be dealt with by defining another function which would iterate over each of the Email bodies extracted and will arrange each of these 5796 individual email bodies as a data frame structure .
The dataframe of all these emails can be visualized as shown in Fig 3 and Fig 4 Fig 5 shows the output in the form a sample extracted email

2.1.2 Checking for missing Values and Empty Emails

As the data set we are dealing with in this project is used, just like typical data it is bound to have some missing values. The dataframe developed earlier consists of our emails, spam and non spam (ham) both so there might be a possibility that out of 5796 emails, there may be some emails which may have no email bodies . During our programming the term missing value for Email bodies can be well defined by either a mail having none value or an email with a string of zero length. The Pandas Dataframe makes it easy to actually check for these two conditions. By accessing the MESSAGE column of our Pandas Dataframe and by using the .isnull().values.any() method we can easily check if there are any none values or null values present in the MESSAGE column . Similarly by accessing the MESSAGE column and using

	MESSAGE	CATEGORY
00001.7848dde101aa985090474a91ec93fcf0	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Tr...	1
00002.d94f1b97e48ed3b553b3508d116e6a09	1) Fight The Risk of Cancer!\n\nhttp://www.adc...	1
00003.2ee33bc6eacdb11f38d052c44819ba6c	1) Fight The Risk of Cancer!\n\nhttp://www.adc...	1
00004.eac8de8d759b7e74154f142194282724	##########...	1
00005.57696a39d7d84318ce497886896bf90d	I thought you might like these:\n\n1) Slim Dow...	1

Figure 2.1: Spam Email Dataframe

	MESSAGE	CATEGORY
00001.7c53336b37003a9286aba55d2945844c	Date: Wed, 21 Aug 2002 10:54:46 -05...	0
00002.9c4069e25e1ef370c078db7ee85ff9ac	Martin A posted:\n\nTassos Papadopoulos, the G...	0
00003.860e3c3cee1b42ead714c5c874fe25f7	Man Threatens Explosion In Moscow \n\n\n\nThur...	0
00004.864220c5b6930b209cc287c361c99af1	Klez: The Virus That Won't Die\n\n \n\nAlready...	0
00005.bf27cdeaf0b8c4647ecd61b1d09da613	> in adding cream to spaghetti carbonara, whi...	0

Figure 2.2: Ham Email Dataframe

the logical condition (data.MESSAGE.str.len() == 0).any() . The False output we receive after applying these checks suggest to us that there are some emails amongst our dataframe which have missing values. Although this does not gives s the exact value of the number of Emails which have missing values

2.1.3 Removing system files

In order to access the files with missing values we can use the logical condition combined with the index locator method of the pandas dataframe to access the emails containing missing values . After looking at the output , we can easily observe that the emails which are having the missing values are just system files. These system files were generated ,the cmds file, that we get into the list is a type of system file that we get into our system after we unzip our data hence this contributes to our empty string or empty email files These files are in our data frame because of our generator function. We can either use the dataframe to either drop of these files or we can manually search for the files in our stored location of our dataset and delete them

2.1.4 Adding Documentation Id to each email in the dataset

After getting rid of the system files, we can now confirm that our dataframe is free of any emails having missing values. Although we still need to modify our Data frame in such a way that it would be easier to comprehend it in a single look. Right now as we can see the first unnamed column of the dataframe is the column of indices for each of our emails in the dataframe. But as we can see the indices are represented by the filenames that each email had when it was downloaded from our dataset resources. So in order to make the dataframe accessible and visually understanding , we will replace these filenames with numbers which will be ranging from 0 to 5795 . These will give us a sense of sequence for all the emails and if we need to access any email from our dataframe we will just use an index to extract a particular email.
Fig 6 gives us an example

DOC_ID	MESSAGE	CATEGORY	FILE_NAME
0	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Tr...	1	00001.7848dde101aa985090474a91ec93fcf0
1	1) Fight The Risk of Cancer!\n\nhttp://www.adc...	1	00002.d94f1b97e48ed3b553b3508d116e6a09
2	1) Fight The Risk of Cancer!\n\nhttp://www.adc...	1	00003.2ee33bc6eacdb11f38d052c44819ba6c
3	##...	1	00004.eac8de8d759b7e74154f142194282724
4	I thought you might like these:\n\n1) Slim Dow...	1	00005.57696a39d7d84318ce497886896bf90d
...	...	...	...
5791	http://news.bbc.co.uk/1/hi/england/2515127.stm...	0	01396.61983fbe6ec43f55fd44e30fce24ffa6
5792	> >-- be careful when using this one.) Also, t...	0	01397.9f9ef4c2a8dc012d80f2ce2d3473d3b7
5793	>>>>> "SM" == Skip Montanaro <skip@pobox.com> ...	0	01398.169b51731fe569f42169ae8f948ec676
5794	So then, "Mark Hammond" <mhammond@skippinet.co...	0	01399.ca6b00b7b341bbde9a9ea3dd6a7bf896
5795	Hi there,\n\n\n\nNow this is probably of no us...	0	01400.f897f0931e461e7b2e964d28e927c35e

Figure 2.3: Dataframe of email with DOC-ID

2.1.5 Visual Representation of Spam and Ham emails in the dataset.

Now that we have arranged and given a sequence to our dataframe we can have a graphical representation of how many spam emails our data frame consists of. It would be handy to have a representation in the form of a pie chart which will give us an overall view of our dataset.

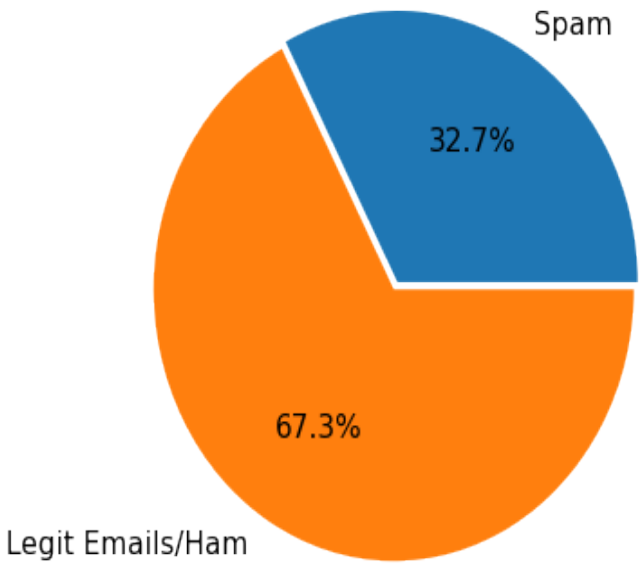


Figure 2.4: Pie-Chart for Spam and Non Spam Emails

2.2 Text Processing techniques

So far as we have seen , if we access a particular mail by its index we can see that email body or the main content consists of HTML Tags , stop words , similar words with same meaning and grammar ,punctuation marks . In this project we will be following the bag of words approach.

Bag of words approach can be summarized as : a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

We will be following this approach in which we will be pre-processing our content of the email body from our dataframe in such a way that the text will disregard grammatical meaning, word order and will get rid of other discontinuities like punctuation , HTML Tags , and stop words.

2.2.1 Converting email body to lower case and Tokenising

Tokenization in simple words is the process of tokenizing or splitting a string , text into a list of tokens . If we access an email from our dataframe the content of our email body is represented as a list of all the content represented as a single string . Before tokenizing we will be converting the entire string of the content of each email to lowercase . We can do so by using the .lower() method of string . In order to perform tokenizing we will be downloading the NLTK resources, this resource will have the “punkt” package which will perform the tokenizing for us. As an example for tokenizing, we will supply a dummy string , which will also be lowercase.

We will be defining a function which will iterate over each email in our dataframe and perform tokenization, lowercasing, removing stop words, removing punctuation marks , Stemming and removing HTML tag altogether.

```
clean_message(data.at[2,"MESSAGE"])
```

```
['fight',
 'risk',
 'cancer',
 'http',
 'slim',
 'guarante',
 'lose',
 'lb',
 'day',
 'http',
 'get',
 'child',
 'support',
 'deserv',
 'free',
 'legal',
 'advic',
 'http',
 'join',
```

Figure 2.5: List of all tokenized word from a single email

2.2.2 Removing Stop Words, Punctuation marks, HTML Tags

In order to train our classifier, we need it to feed and train relevant data , so as to it may be later be able to classify on the testing data which email is spam and which is not spam (ham).In this case relevant data corresponds to words in our email which may indicate that an email is spam or ham. For the purpose of analyzing text data and building our classifier these stopwords might not add much value to the meaning of the document.

Generally, the most common words used in a text are “the”, “is”, “in”, “for”, “where”, “when”, “to”, “at” etc. Consider this text string as an example – “There is a pen on the table”. Now, the words “is”, “a”, “on”, and “the” add no meaning to the statement while parsing it. Whereas words like “there”, “book”, and “table” are the keywords and tell us what the statement is all about. We will be using the NLTK resources that we have downloaded earlier and use the stopwords package provided in it. We will define a set from this stopwords package and when our function will iterate over each email body , the words which will match with this stopwords set will be removed.

After tokenizing , we can observe that , in the list of tokenized words , even punctuation marks are considered as string characters. Punctuation marks do not provide any relevant information as well. Hence we will be required to eliminate the punctuation marks. We can check for punctuation marks in the list using the .isalpha() method for strings.

Looking at the content of our email bodies, we can observe that there are a lot HTML tags , these HTML tags similar to stop words do not provide any value about the email context. HTML Tags from the email body can be eliminated by importing BeautifulSoup , a package from the module bs4 .

2.2.3 Word Stemming

Word Stemming : Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. If we iterate the function we defined for removing stop words, punctuations, HTML tags and to tokenize over each single mail in our dataframe we may end up with words which may deliver the same meaning . For example words like eating and eat provide the same information , the information that is relayed through these words do not contribute much information . Hence the overall purpose of stemming all the words is to reduce a word to its stem so as it conveys the same information. Word Stemming will be performed by a package called PortStemmer which we import from the previously Downloaded NLTK resources. The packages can be varied depending upon the language . Word Stemming can be understood by the example shown in the following figure

```
msg = " All Work And No play makes Jack a dull Boy . To Be or not to BE.????##\
      TheN he goes to fishing beacause he like eating fish and crabs"
```

Figure 2.6: Example string for performing stemming and punctutaion removal

```
['work', 'play', 'make', 'jack', 'dull', 'boy', 'goe', 'fish', 'beacaus', 'like', 'eat', 'fish', 'crab']
```

Figure 2.7: Output of stemming and punctuation removal

In order to perform the techniques mentioned above, a function is defined which would perform all the above techniques and iterate over each one of the Email present in our dataframe The output that we will be receiving after the execution of the function will be a nested listed , containing a separate list corresponding to each email containing pre-processed words.

2.3 Generating Vocabulary

After preprocessing the data that we have , we ended up with a nested list having only lists of individual words as strings. Although if we observe the list of strings we can see that all the words together for a single mail can't form a meaningful sentence. But the words alone are enough to convey the meaning and information. Moving ahead we can create a list of spammy words i.e words that occur more frequently in a spam emails and we can also list out the number of their occurrence as well To do that we create a subset of spam emails and ham emails from our existing dataframe, for spam emails subset we will get a nested list containing tokenized words for each email , that is we will have 1896 lists. From that we can create a pandas series for the spam emails , and then use .value-counts() method for a series. We can then access the top 10 most frequent spam words by using the range of the index

As we can see the top 10 most frequent words occurring in spam emails are as shown above in the fig.

The same can be done in case of Ham emails and we can have the top 10 most frequent words occurring in ham emails are as shown above in the fig.

```
http      3097
email     3090
free      2585
click     2058
receiv    1989
list      1971
get       1914
pleas     1852
busi      1792
order     1746
dtype: int64
```

Figure 2.8: Top 10 most frequent words in spam emails

```
ham_words[:10]
```

```
http      7563
use       3633
list      2880
one       2373
get       2286
mail      2255
would     2003
like      1931
messag    1849
work      1800
dtype: int64
```

Figure 2.9: Top 10 most frequent words in ham emails

Although we are having a list of all the processed words in all the 5796 emails , we won't be using all the words that appear in these emails , we will be using the 2500 most frequent words which appear throughout the 5796 mails. Some application or high end email services also use a vocabulary list which they use for determining and classifying spam emails, although the length of their vocabulary list varies from 50000 words to 100000 words. The reason we are settling down for 2500 words is that we are considering the computational power for our machine. In order to generate a vocabulary list we will pre specify the vocabulary size , and perform the same operations as we did on the spam email subset to get the frequent spam words , but instead of iterating over the spam email subset we will iterate over the whole data frame and use the series functions to get the value counts for each word, but as we have prespecified the vocabulary size we will limit the size of the list to 2500 words. We will create a dataframe containing all these 2500 words and assign an index to it. Then we will also save this Vocabulary dataframe to a CSV file , which can be used later on . This Vocabulary list will further help us to create a sparse matrix for the features.

	VOCAB_WORD
WORD_ID	
0	http
1	use
2	list
3	email
4	get

Figure 2.10: Top 5 Vocabulary words

2.4 Generating Features and dataframe with one word column

As we now have a list of vocabulary we will treat those 2500 words as our features , we will check for the count of their occurrence and can then easily calculate the respective probabilities for each word . Although to proceed further , we will need a proper arrangement of the dataframe we have with us. We will first aim to create a dataframe with one word column , and then we will continue modifying it to a full matrix and then to a sparse matrix. These matrices help us to check or keep a tabular form of the number of occurrences and their occurrences in multiple mails according to the index. Firstly we will try to build a One word column dataframe , as the name suggests this dataframe will have multiple columns and each column will have a single word as its entry. The number of columns this dataframe will have depends upon the length of the longest email in our complete dataset . As we can see in the figure below , the longest email has 7670 words in it, hence the number of columns in our One word column dataframe will be 7670. Each row of this Data Frame will represent a single email. From the image we can see

that row index 0 represents the very first email in our data set and if we keep the row number constant and read over the range of the columns we will end up getting an email body for the first email. The total shape of this Data Frame is (5796, 7671) which represents 5796 emails and 7671 words. It is to be noted that although the last columns may show a none value, there might be some mail in between the range of 5796 mails which may have the end rows having some word.

	0	1	2	3	4	5	6	7	8	9	...	7661	7662	7663	7664	7665	7666	7667	7668	7669	7670
0	save	life	insur	spend	life	quot	save	g	famili	financi	...	None	None	None	None	None	None	None	None	None	None
1	fight	risk	cancer	http	slim	guarante	lose	lb	day	http	...	None	None	None	None	None	None	None	None	None	None
2	fight	risk	cancer	http	slim	guarante	lose	lb	day	http	...	None	None	None	None	None	None	None	None	None	None
3	adult	club	offer	free	membership	instant	access	site	user	name	...	None	None	None	None	None	None	None	None	None	None
4	thought	might	like	slim	guarante	lose	lb	day	http	fight	...	None	None	None	None	None	None	None	None	None	None

5 rows × 7671 columns

Figure 2.11: One Word Column Dataframe

2.5 Splitting Data Into Training set and Test Data set

In order to train and test our classifier, we will need to first train our classifier over a set of training data , and then test its performance on a subset of data which has never been performed before. We will be subsetting the whole dataset of spam and non spam emails into training and test data. As we have seen in images before our dataset had already labeled emails, that it is it was labeled 1 for spam emails and 0 for ham emails. We will be supplying the word column dataframe from the email body that we have pre-processed as Xtrain and Ytrain subsets , and we will further be subsetting the category labels as well into Xtest and Ytest . We will be randomly splitting the Dataset using the train test split module from sklearn to perform the splitting and we will also set a random seed to induce randomness.

We will further analyze the training data subsets and test data subsets. As we can see in the image, our Xtrain training data set has 4057 randomly selected emails and 7670 columns from the One word dataframe.

Similarly our Ytrain training dataset has 4057 rows , with representing spam for 1 and ham for 0

Similarly if we looked into the number of datapoints our test data has and we add the training and test subsets together we will get the original number of 5079 of emails as our output. Implying that the split was successful and had no errors.

```
X_train.head()
```

OC_ID	0	1	2	3	4	5	6	7	8	9	...	7661	7662	7663	7664	7665	7666	7667	7668	7669	7670
4844	ye	inde	agent	directori	verita	cd	unix	subdirectori	file	call	...	None	None	None	None	None	None	None	None	None	None
4727	problem	come	tri	instal	harddissssk	like	alreadi	mount	http	yahoo	...	None	None	None	None	None	None	None	None	None	None
5022	origin	messag	date	mon	aug	chad	norwood	sven	cc	subject	...	None	None	None	None	None	None	None	None	None	None
3504	inlin	folk	sever	major	internet	outag	morn	across	major	provid	...	None	None	None	None	None	None	None	None	None	None
3921	url	http	date	bath	chronicl	None	None	None	None	None	...	None	None	None	None	None	None	None	None	None	None

Figure 2.12: Xtrain Dataset

```
Y_train.head()
```

```
DOC_ID
4844    0
4727    0
5022    0
3504    0
3921    0
Name: CATEGORY, dtype: int64
```

Figure 2.13: YTrain Dataset

Chapter 3

Training The Naive Bayes Classifier

3.1 Analysing the Sparse Matrix for Training and Test dataset

"Now that we have segregated our dataset into training and testing data sets , we will be focusing on the training data to create a sparse matrix , this sparse matrix will be crucial to train our Classifier and it will help us to check the occurrences of our pre-processed words or tokens and our pre-sorted vocabulary list. In order to create a sparse matrix , first we will create an indexed object of our vocabulary list, each word on this indexed object will help us get an index when a word from the vocabulary is specified, the index will help locate our word (token) from a dataframe. We will define a function to build a sparse matrix of all the words in the emails present in the training data set Xtrain , the function will take the parameters Xtrain as a dataframe , our indexed vocab and Ytrain as the labels data. The Sparse Matrix generated will look like shown in the figure below

	LABEL	DOC_ID	WORD_ID	OCCURENCE
0	0	4844	265	1
1	0	4844	1239	1
2	0	4844	504	1
3	0	4844	308	1
4	0	4844	254	1

Figure 3.1: Sparse Matrix for Training Dataset

Understanding Sparse Matrix :

- Each row in the Sparse matrix is indexed , this index serves well for identification and axis.
- The first column represents the label category of each email which will be either 0 or 1 (0- for Ham Emails and 1- for Spam Emails)
- The second column represents the DOC-ID , the number we assigned each email from the original dataset for identification and accessing. The DOC-ID 4844 refers to the 48843rd Email in our original dataset (as the range started from 0)

- The third column represents the Word-ID, the numbers in this column represents a specific word which is present in our indexed vocabulary list.
- And the last column represents the occurrence column , it holds the record for the occurrence of the specific Word-id in that specified DOC-ID email.

The current sparse matrix will give a single occurrence for each token or word , even if a word is repeated in any of the emails , it will create a duplicate Word-ID for the same word and it will display the occurrence as 1. But this is not useful for us , we need a sparse matrix which will give us an exact idea about the occurrence of each word by its Word-ID. To achieve that we use the groupby() method of the pandas dataframe and apply it on sparse matrix dataframe .This results in a sparse matrix which gives us an exact occurrence count for each word represented by its Word-ID
Note : This sparse matrix is for training data

	DOC_ID	WORD_ID	LABEL	OCCURENCE
0	0	2	1	1
1	0	3	1	2
2	0	4	1	1
3	0	7	1	3
4	0	11	1	1

Figure 3.2: Grouped Sparse Matrix for Training Data

The above figure represents a grouped sparse matrix for training data which gives us the count of the occurences for the words represented by Word-ID 2, 3, 4, 7, 11 for the email with DOC-ID 0 i.e the very first mail in our dataset

We will carry out the same procedure for creating a sparse matrix for the test data as well . And we will save both these grouped sparse matrix data frames into our system as text files.

	DOC_ID	WORD_ID	LABEL	OCCURENCE
0	0	2	1	1
1	0	3	1	2
2	0	4	1	1
3	0	7	1	3
4	0	11	1	1

Figure 3.3: Grouped Sparse Matrix for Test Data

3.2 Full Matrix Generation (for summing the occurences of a word)

After saving the sparse matrix for both training and test data as text files , we will use the numpy module to load them into separate variables as arrays.Firstly we will work with the training data grouped sparse matrix , we will load it as an array using numpy. This training data array will have 258332 rows and as it is formed by the training data set it will have 4014 emails. Now we will create an empty data frame , empty as in each entry in this dataframe will have zeroes .This dataframe will be created in such a way that it has 258332 and the number of columns will be labeled with range from 0 to 2499 representing the 2500 words present in our vocabulary. We will then define a function which will take inputs from the grouped sparse matrix for training data and iterate over this empty dataframe to fill up the entries for the words and count the number of occurrences per email. For each iteration the occurrence of each word in the vocabulary will be updated and we will end up with a Full Matrix.

Understanding the Full Matrix :

- Each column in this Full Matrix represents the Word-ID for a particular word in the vocabulary list. As there are 2500 words predefined in our vocabulary list, there will be columns ranging from 0 t0 2499
- As there are 4014 emails in the training dataset , we will have 4013 rows each corresponding to an email in the training dataset
- And each entry in a column with respect to a specific row represents the number of occurrences of the word with that Word-ID for that particular email with that $\mathrm{DOC}_I D$.

DOC_ID	CATEGORY	0	1	2	3	4	5	6	7	8	...	2490	2491	2492	2493	2494	2495	2496	2497	2498	2499
5789	0	3	1	0	1	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
5790	0	1	1	1	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
5791	0	3	1	0	1	1	0	0	0	1	...	1	0	0	0	0	0	0	0	0	0
5794	0	1	1	1	0	0	1	2	0	0	...	0	0	0	0	0	0	0	0	0	0
5795	0	3	4	2	0	5	0	3	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 3.4: Full Matrix for Test Data

3.3 Calculating Probabilites and Smoothing

Now that we have the full matrix which gives us an exact idea about the occurrence of words or tokens present in our vocabulary over our training dataset, we can now work out the probabilities for each token or word in the Vocabulary containing 2500 words.

Consider for example we want to calculate the probability of an email being spam given that it contains a specific word like “Viagra” The formula will be Insert Formula

$$P(Spam \mid Viagra) = \frac{P(Viagra \mid Spam)\, P(Spam)}{P(Viagra)}$$

To calculate this probability we will be needing :

- The overall probability of an email being spam : P(Spam)
- The probability of the word "Viagra" appearing in any email - P(Viagra)
- The probability that an email containing the word "Viagra" given it is a spam email.$P(Viagra\,|Spam)$
- If we make a direct comparison and assume that "Viagra" is one of the words in our vocabulary , then we can Calculate P(Viagra-spam) by
- (No of occurrence of the word Viagra in spam emails) (No of words in spam emails) Calculate P(Viagra) by
- (Total number of times the word "Viagra occurs in both spam emails and ham emails) (Total number of words across the dataset both spam emails and ham emails)

With the help of the Full Matrix from training dataset and python code we will be able to achieve a few of these numbers to calculate the probabilities for each word in our vocabulary. This acts as the training step for our Naive Bayes Classifier. The overall probability of spam which we assume for the training data set can be calculated by accessing the size of the overall emails in the training dataset which comes out to be 4014 and then dividing it by the total number of spam emails present in our dataset. The probability comes out to be 31

```
prob_spam = full_train_data.CATEGORY.sum() / full_train_data.CATEGORY.size
print('Probability of spam is', prob_spam)
```

```
Probability of spam is 0.310989284824321
```

Figure 3.5: Probability of spam

To get a hold of the total number of words or tokens (only words which are in our vocabulary) per email we will make use of the full matrix and sum up all the occurrence of each words column by column for each row , for this we will make use of the sum () function to our Full matrix data frame and provide it a n argument called axis =1 which will perform the sum one column by another column We will store the token or word count per email as a series and if we take an overall sum of this series we will get a sum of all the vocabulary words appearing in the overall 4104 emails. i.e No of words or tokens throughout the training dataset. The figure below explains the process and our output.

Similarly we will use our code to find the number of tokens present in the overall spam emails present in the training data and we will also find the number of tokens present in the overall ham emails.

The full train matrix that we have has 4104 rows and 2500 columns .We will create a subset of this full train matrix which will only have spam emails out of the total 4014 emails.The subset of this dataframe will have 1249 rows as it will contain 1249 spam emails. The fig 3.9 represents the first 5 rows of this subset.Now we will sum up all the occurrences that occur for each token in this subset over all the 1249 rows and represent it as a pandas series. This series will give

```
email_lengths[:5]
```

```
DOC_ID
0     87
1     53
2     40
3    183
4     43
dtype: int64
```

```
total_wc = email_lengths.sum()
total_wc
```

```
429184
```

Figure 3.6: tokens per emails and overall sum of tokens over the training dataset

```
spam_wc = spam_lengths.sum()
spam_wc
```

```
176318
```

Figure 3.7: word count for spam emails(training)

```
nonspam_wc = ham_lengths.sum()
nonspam_wc
```

```
252866
```

Figure 3.8: word count for spam emails(training)

DOC_ID	0	1	2	3	4	5	6	7	8	9	...	2490	2491	2492	2493	2494	2495	2496	2497	2498	2499
0	0	0	1	2	1	0	0	3	0	0	...	0	0	0	0	0	0	0	0	0	0
1	7	1	2	0	1	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
2	6	1	1	0	1	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
3	6	0	0	2	4	0	3	14	0	0	...	0	0	0	0	0	0	0	0	0	0
4	5	1	2	0	1	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 3.9: first five rows for the full matrix only for spam

```
summed_spam_tokens.head()

0    2179
1     935
2    1217
3    2022
4    1219
dtype: int64
```

Figure 3.10: series representing occurence of tokens overall spam emails

us the total count of each token across the overall spam emails. Fig 3.10 shows the particular pandas series reflecting that.

While calculating the sum of occurrences we would be performing smoothing in order to avoid zero division. We perform smoothing by adding 1 to the total while iterating over the tokens, just in case any taken had zero occurrences.

We will also perform the same procedure and calculate the overall occurrence of tokens over the complete ham email subset and get a series of total occurrences of each token

Now as per the formula for bayes theorem$P(Spam\ |Token)$ we will be required to calculate 3 probabilities:

- $P(Token\ |Spam)$ - Probability that a Token Occurs given the Email is Spam

P(Token | Spam) - Probability that a Token Occurs given the Email is Spam

```
prob_tokens_spam = summed_spam_tokens / (spam_wc + VOCAB_SIZE)
prob_tokens_spam[:5]

0    0.012186
1    0.005229
2    0.006806
3    0.011308
4    0.006817
dtype: float64
```

Figure 3.11: series representing probabilities of tokens given spam emails

- $P(Token\ |Ham)$- Probability that a Token Occurs given the Email is Ham
- P(Token) - Probability that a Token Occurs

Now that we have a series representing the occurrence of token overs spam and ham emails , we can get an overall count of a certain token over a set of spam emails and ham emails .while calculating $P(Token\ |Spam)$ and $P(Token\ |Ham)$ we will account for smoothing and add vocabulary size to the denominator

Once we have a series of all these three probabilities , we will save them to text files using numpy for further use.

P(Token | Ham) - Probability that a Token Occurs given the Email is Nonspam

```
prob_tokens_nonspam = summed_ham_tokens / (nonspam_wc + VOCAB_SIZE)
prob_tokens_nonspam[:5]
```

```
0    0.021475
1    0.010142
2    0.008008
3    0.003673
4    0.006313
dtype: float64
```

Figure 3.12: series representing probabilities of tokens given non spam emails

P(Token) - Probability that Token Occurs

```
prob_tokens_all = full_train_features.sum(axis=0) / total_wc
```

```
prob_tokens_all[:5]
```

```
0    0.017850
1    0.008209
2    0.007596
3    0.006892
4    0.006592
dtype: float64
```

Figure 3.13: series representing probabilities of tokens over all emails

We will do the same with the test data as well , from our earlier process we had obtained a text file for the sparse test data matrix , we will now apply the same full matrix function over the test data that we have i.e X-test and Y-test . After applying the function we can store the full Matrix in a variable and then subset it again into X-test holding the occurrences of the full Matrix and Y-test holding on to labels and categories. We will save both of these test data entries into text files which can be used later.

X_test

DOC_ID	0	1	2	3	4	5	6	7	8	9	...	2490	2491	2492	2493	2494	2495	2496	2497	2498	2499
8	0	0	1	4	2	1	2	4	1	2	...	0	0	0	0	0	0	0	0	0	0
12	6	1	1	0	1	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
14	0	0	1	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
15	0	2	1	1	2	0	0	3	0	4	...	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	1	0	1	0	0	0	...	0	0	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5783	2	1	0	2	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5786	5	5	2	2	1	2	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
5788	0	4	0	2	4	3	3	1	4	3	...	0	0	0	0	0	0	0	0	0	0
5792	2	2	0	1	0	0	2	1	0	0	...	0	0	0	0	0	0	0	0	0	0
5793	1	9	1	0	0	1	2	1	1	1	...	0	0	0	0	0	0	0	0	0	0

Figure 3.14: X-test data full matrix

```
y_test
DOC_ID
8          1
12         1
14         1
15         1
17         1
         ..
5783       0
5786       0
5788       0
5792       0
5793       0
Name: CATEGORY, Length: 1724, dtype: int64
```

Figure 3.15: y-test data category series

Chapter 4

Testing and Evaluation of Classifier

4.1 Joint Conditional Probabilities (Prior and Dot Product)

To begin with the testing we will first load all the text files into variables which can be later used. The files we will be referring to will contain.

- The data in the form of probability that a Token occurs given the Email is Spam
- The data in the form of probability that a Token occurs given the Email is Ham
- The data in the form of probability that a Token occurs in all the emails
- The data in the form of Full Matrix for test data i.e X-test
- The data in the form of Labels and categories i.e Y-test

In Naive Bayes Algorithm we assume that all the events happening are independent of each other .Hence the Joint Probability for two independent events A and B is given by

$$P(A \cap B) = P(A) \times P(B)$$

So in the case of our classifier , Let's say for example we get an email which says
"Hello Friend want some Free Viagra "
Then the probability that this mail is spam , by Naive Bayes Theorem , is given by

$$P(Spam \,|\, Hello) \times P(Spam \,|\, Friend) \times P(Spam \,|\, Free) \times P(Spam \,|\, Viagra)$$

Where Hello , Friend, Free, Viagra all are words belonging to our vocabulary list . Now in relation to the tokens , we have calculated some probabilities for the tokens , although just like the example above we haven't multiplied the token probabilities altogether with the X-test , that is our Full test matrix . On a closer look we can see that in the below fig that our X-test is an array with thousands of rows and columns . To adapt Bayes theorem and to deal with all these tokens at the same time we use Dot product for multiplying the X-test array with our three probabilities. To show it in a more mathematical way we will have

$$P(Spam \,|\, Tokens) = X\_test \cdot \frac{P(Tokens \,|\, Spam)\, P(Spam)}{P(Token)}$$

We will carry out the dot product by using the dot method of Numpy , firstly we will calculate the dot product of the X-test array and P (Tokens $\|Spam$)

Now that we are working with the test data , we will have to work according to the formula given by the Bayes Theorem . P (Spam) will be a prior probability which we will be making a guess based on our previous training data and will set it to 31 According to Bayes theorem the formula for spam and non spam emails is given by

$$P(Spam \mid X) = \frac{P(X \mid Spam)\, P(Spam)}{P(X)}$$

$$P(Ham \mid X) = \frac{P(X \mid Ham)\, (1 - P(Spam))}{P(X)}$$

Also we will be modifying the formula a bit , and will take log of all the probabilities we have , this is done just to make calculations a bit more easier. As mentioned earlier , in order to get the overallJoint probability $P(Spam \mid Tokens)$, which represents probability of an email being spam given all tokens are present in it , will be given by multiplying the array of X-test with the modified formula we have , which is as shown in the figure.

```
joint_log_spam = X_test.dot(np.log(prob_token_spam) - np.log(prob_all_tokens)) + np.log(PROB_SPAM)
```

```
joint_log_spam[:5]
```

```
array([24.27580867,  2.15999942, 20.59075715, 17.74776262, 20.50984493])
```

Figure 4.1: Joint Probability modified formula for spam

```
joint_log_ham = X_test.dot(np.log(prob_token_ham) - np.log(prob_all_tokens)) + np.log(1-PROB_SPAM)
```

```
joint_log_ham[:5]
```

```
array([-60.96824846, -11.00905316, -37.9678354 , -59.13066195,
       -53.80247296])
```

Figure 4.2: Joint Probability modified formula for Ham

Now that we have an array of joint log probabilities for tokens for spam emails and non spam emails , we can go ahead and make predictions .

4.2 Comparing Joint Probabilities

"Our classifier will make predictions based on a simple check over the arrays of probabilities that we have of our test data.It will compare the joint probabilities between spam and non spam emails and would classify the mails to spam or non spam while satisfying the below condition.

$$P(Spam \mid X) > P(Ham \mid X)$$

OR

¡

$$P(Spam \mid X) < P(Ham \mid X)$$

If the $P(Spam \mid, X)$ is higher then the mail would be classified as a Spam email. if the $P(Ham \mid, X)$ is higher then the mail would be classified as a Ham email.

Using the above condition , we make comparisons with the joint probabilities of spam and non spam emails and save the predictions in a variable . In the figure below , we can see that , the predictions that we made for the last 10 emails in our test data set show that the last 10 emails contain 9 non spam emails and 1 spam email .

```
prediction = joint_log_spam > joint_log_ham
```

```
prediction[-10:]*1
```

```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
```

Figure 4.3: Predictions for the last 10 emails in the testing data set

4.3 Metrics

We have built our Naive Bayes Classifier from scratch and we have trained it on our training data and we have classified our emails on our training data ,but we have not evaluated the performance of our model , we don't know yet if our model is doing a good job in classifying or not . The basic criteria for judging our classifier is that if a mail is spam it should classify it as spam and if it is legitimate it should classify it as Ham or normal mail. Metrics we will be looking into below help us understand the performance of our model.

4.3.1 The Accuracy Matrix

The Accuracy metrics give us an idea about how many emails did our classifier classified correctly and how many emails did your model classified incorrectly. In order to check for the accuracy of our model we will be needing to calculate two things

- Number of emails that were classified correctly
- Number of emails that were not classified correctly

We can do this by setting up a logical condition as shown in the figure below and get both of our required point.

```
correct_docs = (y_test == prediction).sum()
print('Docs classified correctly', correct_docs)
numdocs_wrong = X_test.shape[0] - correct_docs
print('Docs classified incorrectly', numdocs_wrong)
```

```
Docs classified correctly 1685
Docs classified incorrectly 39
```

Figure 4.4: Number of emails classified correctly and incorrectly

There are two ways to check the accuracy of our model , the first way is to use the corrected number of docs and divide the number by total number of emails present in the test dataset. The other way is to use the incorrectly classified emails and follow the same procedure, but at the end subtract the output from 100 , which will give you the exact accuracy for the model

```
# Accuracy
correct_docs/len(X_test)
```

```
0.9773781902552204
```

Figure 4.5: Method I

```
fraction_wrong = numdocs_wrong/len(X_test)
print('Fraction classified incorrectly is {:.2%}'.format(fraction_wrong))
print('Accuracy of the model is {:.2%}'.format(1-fraction_wrong))
```

```
Fraction classified incorrectly is 2.26%
Accuracy of the model is 97.74%
```

Figure 4.6: Method II

4.3.2 Visualising the Decision Boundaries

"We will be visualizing our results for the predicted emails, we will be plotting the joint conditional probability for emails which are spam vs Joint conditional probability for emails which are legitimate . We will be using matplotlib and seaborn libraries to help us plot these visuals.we will be plotting these joint conditional probabilities by using the scatterplot method , so that we end up getting a scatterplot figure. Looking back at our prediction condition, our algorithm decided if a mail is spam or ham , based on which probability was greater. To understand our visualizations better we will be using a decision boundary , the decision boundary will divide the plot space in to two specific regions being spam and non -spam respectively . If a point lies on to the top most left corner of the plot , we can consider that it had a high probability of being spam and has a low probability of being ham , the vice versa also supports this logic. Following up the same logic, we can understand that our decision boundary will be a diagonal line starting from the origin and ending at the right hand corner , points below it will be referred to as ham emails and points above it will be referred as spam emails

- The data points are mostly populated at the right corner , which makes us difficult to understand the classification of our emails , this is an issue which arised due to overplotting.
- The second problem we are facing is that we can make out from the plot which points were correctly classified and which were not correctly classified.

To overcome these issues

- Firstly we will include some transparency for the data points
- We can make the data points smaller by reducing their size
- We can create a subplot of this original plot , the subplot will be mostly focusing on the top right corner giving us a clear view.

We can also plot a zoomed in version of the scatterplot by using Seaborn module.The figure below shows us the Decision Boundary and a clear visually appealing segregation of classified emails.

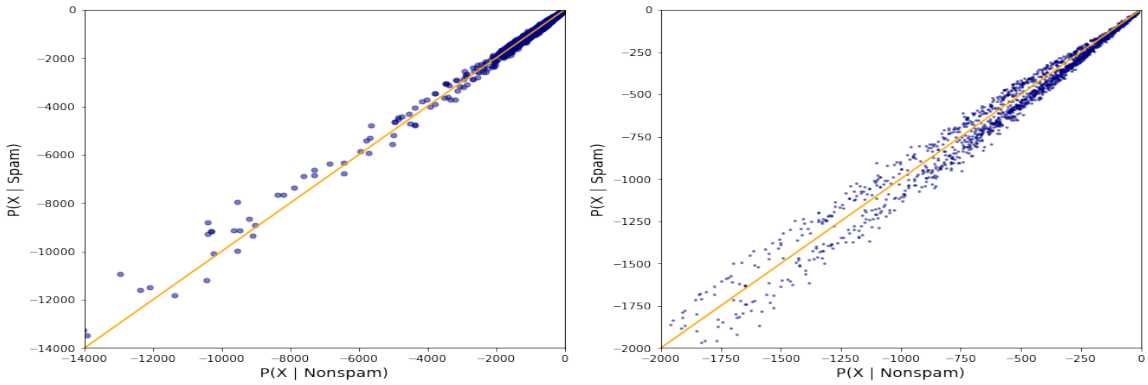


Figure 4.7: Scatterplot for classified emails with Decision Boundary using Seaborn

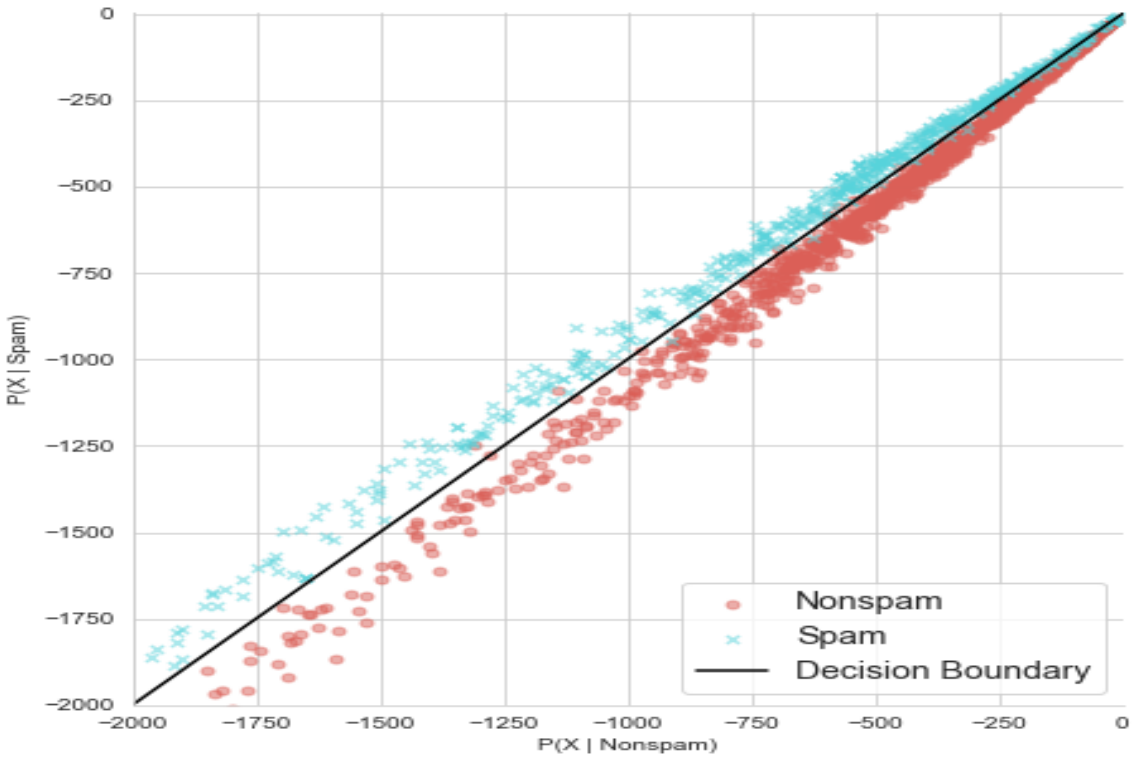


Figure 4.8: Scatterplot for classified emails with Decision Boundary using Seaborn

4.3.3 False Positives vs False Negatives

There might be some cases where it won't be advantageous to select a model solely based on the Accuracy Metric . Hence it is always preferred to look into other Performance metrics as well. In context to our classifier , False positive will be a case where our classifier will predict that it is a Spam email , but in reality it is a Non -Spam email. False Negative is where are spam filter classified a spam email as a Non-Spam Email, With respect to our project , we use the unique method from our numpy module with the combination of logical conditions to get true positive value , false positive value and false negative value.

```
np.unique(prediction, return_counts=True)
(array([False,  True]), array([1136,  588], dtype=int64))

true_pos = (y_test == 1) & (prediction == 1)

true_pos.sum()
569
```

Figure 4.9: True Positive value

```
false_pos = (y_test == 0) & (prediction == 1)
false_pos.sum()
19

false_neg = (y_test == 1) & (prediction == 0)
false_neg.sum()
20
```

Figure 4.10: False Positive and False Negative Values

4.3.4 Recall Metric

"Recall score is also known as Sensitivity . Recall is defined by a formula which is given by

$$Recall = \frac{True\,Positive}{True\,Positive + False\,Negative}$$

If we look into the formula it is to be observed that if False Negative decreases the Recall score increase . Also if the False Negative value increases the Recall score decreases.The maximum value of Recall goes to 1. For our project , Recall can be interpreted by asking a simple question , out of all the spam emails, how many of these we actually identified as spam emails. The figure below shows us the Recall Score for our project

```
recall_score = true_pos.sum() / (true_pos.sum() + false_neg.sum())
print('Recall score is {:.2%}'.format(recall_score))

Recall score is 96.60%
```

Figure 4.11: Recall Score

4.3.5 Precison Metric

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Positives}$$

Precision can be defined as the ratio of correctly predicted Spam messages to the total number of times we predicted an email was spam. The ratio looks very similar to recall but instead of False Negative it uses False Positives in the denominator. By observing the formula we can say that a high Precision equals to low False Positives rate . In our project , False Positives correspond to an email which is a non - spam message which ends up classified as a spam message.Precision can be considered as a measure of quality. The fig below shows us the Precision value for our classifier for spam and non spam emails

```
precision_score = true_pos.sum() / (true_pos.sum() + false_pos.sum())
print('Precision score is {:.3}'.format(precision_score))
```

```
Precision score is 0.968
```

Figure 4.12: Precision

4.3.6 F1-Metric

$$Recall = 2 \times \frac{Precision \times Recall}{Precion + Recall}$$

In many cases we need High Recall Score and High Precision , both of these metrics can be combined into a single metric called the F1 score. The formula tells us that if we need a balanced classifier we will need to maximize both the Recall score and Precision. The above formula gives us the F1 score. The figure below shows us the F-1 score for our classifier.

```
f1_score = 2 * (precision_score * recall_score) / (precision_score + recall_score)
print('F Score is {:.2}'.format(f1_score))
```

```
F Score is 0.97
```

Figure 4.13: F-1 Score

Chapter 5

Summary and Conclusion

5.0.1 Summary

The over all project aim was to creat a spam filet based on Naive Bayes ALgorith from scratch . Although there are many such projects available which make use of libraries instead of building it from scratch , it helped us to understand the exact working and application of the Naive Bayes Algorithm. As such although the performance matrix for this classfier are quite satisfactory , there are still some areas where the algorithm can be improved. The overall use of dataframes was made only possible because of pandas library and the visualisation was carried out by using the matplotlib as well as seaborn library from python . For further uses , the size of the data set can be increased and more functionality can be added to this project if there is enough computational force supporting the claim.

5.0.2 Conclusion

E-mail spam filtering is an important issue in the network security and machine learning techniques; Na¨ıve Bayes classifier that used has a very important role in this process of filtering e-mail spam. The quality of performance Na¨ıve Bayes classifier is also based on datasets that used. As can see, dataset that have fewer instances of e-mails and attributes can give good performance for Na¨ıve Bayes classifier. Na¨ıve Bayes classifier also can get highest precision that give highest percentage spam message manage to block if the dataset collect from single e-mail accounts. So we can see, why performance of Na¨ıve Bayes classifier is good when used SPAMBASE dataset.

Chapter 6

References

1. Notes on Naive Bayes Classifiers for Spam Filtering by Jonathan Lee 9School of Computer Science and Engineering, University of Washington)

2. Analysis of Naive Bayes Algorithm for Email Spam Filtering across Multiple Datasets Nurul Fitriah Rusland, Norfaradilla Wahid, Shahreen Kasim, Hanayanti Hafit Department of Web Technology, Faculty of Computer Science and Information Technology, Universiti Tun Hussein On Malaysia