Email Classification and Meeting Scheduling Using Classifier Algorithm

By

Behroz Newaz Khan – 12101023
Sk Golam Saroar – 13101251
Md. Mosfaiul Alam – 13101047
Sebastian Romy Gomes – 13101058

Supervisor
Dr. Amitabha Chakrabarty

Co-Supervisor
Mr. Moin Mostakim

Department of Computer Science and Engineering
BRAC University

Thesis report submitted to BRAC University in accordance with the requirements for the degree of Bachelor of Science in Computer Science & Engineering

Submitted in April 2017

Declaration

We, hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researchers are mentioned by reference. This thesis, neither in whole or in part, has been previously submitted for any degree.

Signature of the Supervisor	Signature of Author
______________________	______________________
Dr. Amitabha Chakrabarty	Sebastian Romy Gomes
Signature of the Co-Supervisor	Signature of Author
______________________	______________________
Mr Moin Mostakim	Sk Golam Saroar
	Signature of Author
	______________________
	Behroz Newaz Khan
	Signature of Author
	______________________
	Mosfaiul Alam Telot

Acknowledgement

Our heartfelt gratitude to our thesis supervisor Dr. Amitabha Chakrabarty and to our co-supervisor Mr. Moin Mostakim for their generous guidance and continuous support throughout our work. Without their help and assistance, we wouldn't be able to finish our research successfully.

We are extremely thankful to our parents, family members and friends for their support and encouragement. The journey would be much harder if they weren't present for us in every moment whenever we needed them.

We would also like to take this opportunity to thank Mr. Touhid Hossain, Mr. Kazi Rezaul Karim – our Lab Technical Officers and Mr. Abdul Karim, Office Assistant who made sure we got all the technical support we needed.

Finally we thank BRAC University for giving us the opportunity to use their resources and complete our thesis in the university.

Abstract

This research investigates a comparison between two different approaches for classifying emails based on their categories. Naive Bayes and Hidden Markov Model (HMM), two different machine learning algorithms, both have been used for detecting whether an email is important or spam. Naive Bayes Classifier is based on conditional probabilities. It is fast and works great with small dataset. It considers independent words as a feature. HMM is a generative, probabilistic model that provides us with distribution over the sequences of observations. HMMs can handle inputs of variable length and help programs come to the most likely decision, based on both previous decisions and current data. Various combinations of NLP techniques- stopwords removing, stemming, lemmatizing have been tried on both the algorithms to inspect the differences in accuracy as well as to find the best method among them. Along with classifying emails, this paper also describes the methodologies used for automatic meeting scheduling by an intelligent email assistant. Users who regularly send or receive messages for setting up meetings will be greatly benefitted by this system as it will classify their emails and schedule their meetings automatically.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

HMM – Hidden Markov Model

POS – Parts of Speech

NLP – Natural Language Processing

CRF – Conditional Random Field

NLTK – Natural Language Toolkit

TP – True Positive

FN – False Negative

TN – True Negative

FP -- False Positive

API -- Application Program Interface

SSL – Secure Sockets Layer

IMAP – Internet Message Access Protocol

PEM -- Privacy-enhanced Electronic Mail

CHAPTER 1

Introduction

Email is one of the most important means of communication in today's world. It creates a fast, reliable form of communication that is free and easily accessible. It is not characterized by the inconveniences that are generally associated with traditional communication media, such as telephone or postal mail. For these reasons, email usage has increased substantially around the world. In 2015, the number of emails sent and received per day totaled over 205 billion. This figure is expected to grow at an average annual rate of 3% over the next four years, reaching over 246 billion by the end of 2019[1]. With increasing use of email, maintaining all these emails has become very essential. A lot of these emails are spam emails. As of December 2016, spam messages accounted for 61.66 percent of email traffic worldwide[2]. Kaspersky Lab figures show that spam email messages containing malicious attachments – malware, ransom ware, malicious macros, and JavaScript – started to increase in December 2015. That rise has continued, and in March 2016 malicious spam email volume had risen to four times the level seen in 2015. In March, 2016, Kaspersky Lab detected 22,890,956 malicious spam emails. Spam email volume as a whole increased over the quarter, rising to an average of 56.92%

1 http://www.radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf

2 https://www.statista.com/statistics/420391/spam-email-traffic-share/

for the first three months of 2016[2]. Spam emails are commercial in nature but may also contain disguised links that appear to be for familiar websites but in fact lead to phishing[3] web sites or sites that are hosting malware. Spam email may also include malware as scripts or other executable file attachments. Therefore, filtering these spam emails has become a crying need for email users around the globe. Most email service providers like Gmail, Yahoo come with spam filter but there are many startups who want to have their own mail servers. They are less likely to have spam filter in those servers. In this report, we have described the methodologies that can be used to classify emails into different categories like important and spam. Also, email use continues to see strong use in the business world, as well as among consumers. A lot of appointments are scheduled over email messages every day. We have to spend our valuable time dealing with the back and forth to get these meetings scheduled. In this paper, we have also demonstrated how an AI powered email assistant can be built to automatically schedule our meetings.

1.1 Research Goals

The goal of our research is to classify email messages and detect whether an email is important or spam using different machine learning[4] algorithms and to schedule meetings automatically if there is a request for meeting in the email. Firstly, our system will categorize the emails into appropriate classes. Then, it will search among the important emails to check if any of those emails are meeting requests. If it identifies an email where a meeting is requested, it will analyze the email to retrieve the date and time for that meeting and will intelligently act to set up the meeting. Classification will be done by observing texts of the email body. In this research, we consider relative words or sentences as feature to classify email messages. We will try to find out the method that gives most accurate classification. There are many machine learning algorithms. Naive Bayes and HMM are two among them. We will use Naive Bayes as it considers independent words as a feature. On the other hand, HMM is a probabilistic graphical model. We will use this

[3] phishing websites: A website that tries to steal your account password or other confidential information by tricking you into believing you're on a legitimate website

[4] See Appendix A.1

model because it denotes the conditional dependency between random variables. The difference in nature between these algorithms makes it interesting to compare them. In our research, we will collect dataset, pre-process them, create models, estimate parameters and evaluate accuracy, precision, recall, f-metrics for both the algorithms. We will use stemming, lemmatizing, removal of stopwords in various combinations with the algorithms to analyze which algorithm on what combination gives us the best result. We will use the better algorithm for email classification.

1.2 Motivation

As a research group, we wanted to do our undergraduate thesis on a research that will assist a large section of people on a daily basis. Email classification and automatic meeting scheduling is a necessity that can save a significant amount of our time every day so that we can focus on more important tasks. All free email services that we use today offer various types of classification. Various algorithms also vary in performance. The approach that we chose deals with Naive Bayes and HMM. Naive Bayes Classifier is based on conditional probabilities. It is fast and works great with small dataset. Bayesian methods can do inference[5] in all kinds of cases where no other method can help. When creating an engineered system, you build a model of the world and then find a good controller in that model. Bayesian methods interpolate to this extreme. The reasons behind choosing HMM is it is a generative, probabilistic model that can handle inputs of variable length. It provides us with distribution over the sequences of observations. HMMs are probability models that help programs come to the most likely decision, based on both previous decisions (like previously recognized words in a sentence) and current data. When we compared these two algorithms- HMM outperformed Naive Bayes and thus encouraged us to choose this algorithm for classifying. We also thought email clients who get huge number of emails containing meeting request need an intelligent email agent who will not only classify their emails accurately but also find the emails with meeting request, extract relevant information and set up the meeting or continue conversation if client’s current schedule clashes with meeting time. This is why we worked on the meeting scheduling algorithm as an extension of classifying emails.

[5] Inference: A conclusion reached on the basis of evidence and reasoning

1.3 Methodology

Whenever our email assistant receives a new email, our classifier algorithm is run on that email. For classifying an email into categories like spam and important, we used machine learning algorithm. We conducted a comparison between Naive Bayes and HMM and found out that HMM gives better result. We later experimented with eight different variants of text processing to increase accuracy where stemming [13] words proved to be the best method. After text processing, only the essential words from that email are left to be compared with our pre-classified knowledge set. HMM algorithm is run and the email is classified into appropriate category. If the category is 'important', then it is a potential candidate for being a meeting related email. Each word of the email is then carefully tagged with proper pos-tag and regular expression is used to pick the sentences that contains relevant information about a meeting scheduling [14]. We have removed stopwords and used chunking to analyze regexp syntax and pull out information like time and location of the meeting. The system fetches user's calendar events using Google calendar API and checks if he has any free slot available on that particular time. [15] After setting the event, the system sends a confirmation email to the user.

1.4 Outline

Chapter 2, describes the background research and basic review about the topic.
Chapter 3, describes the terminology about what classification is. It also describes the NLTK toolkit, stemming, lemmatizing, stopwords and chunking. Sheds light on the algorithms, dataset, and system setup used in this research.
Chapter 4, describes the methodologies.
Chapter 5, demonstrates the analysis and result.
Chapter 6, describes limitations and future scope of the research along with conclusion.

CHAPTER 2

Literature Review

Email classification can be applied to several different applications, including filtering messages based on priority, assigning messages to user-created folders, or identifying spam. We will focus on distinguishing important emails from spam emails. One major consideration in the categorization is that of how to represent the messages. Specifically, one must decide which features to use, and how to apply those features to the categorization. M. Aery et al. [1] gave an approach which is based on the premise that patterns can be extracted from a pre-classified email folder and the same can be used effectively for classifying incoming emails. Since emails exhibit a structure in the form of headers and the message body, the relationships between various terms (e.g., the occurrence of a term in the subject or body of the message) can be represented in the form of a graph. They have chosen graph mining as a viable technique for pattern extraction and classification. R. Islam et al. [2] gave an approach which proposed a multi-stage classification technique using different popular learning algorithms with an analyzer which reduces the FP[6] problems substantially and increases classification accuracy compared to similar existing techniques. X. Wang et al. gave an approach [3] which reviews recent approaches to filter

[6] FP (False Positive) is the proportion of negative cases that were incorrectly classified as positive, as calculated using the equation: FP = b/(a+b).

out spam email, to categorize email into a hierarchy of folders, and to automatically determine the tasks required in response to an email. B. Klimt et al [4] gave an approach that introduced Enron corpus as a new dataset for this domain. V. Bhat et al. [5] gave an approach which derives spam filter called Beaks. They classify emails into spam and non-spam. Their pre-processing technique is designed to identify tag-of-spam words relevant to the dataset.
There are two main methods for detecting spam email that are widely used. One is sender based spam detection and the other method is content based spam detection which will consider only the content of an email. The distinction between sender based spam detection and content based spam detection is the options that are used for classification. In sender based detection, the email sender information such as the writing style and the email sender user name is used as the major features E.Yitagesu1 et al [6]. In content based detection, terms extracted from the emails are the major features. The research paper written by S.Teli [7] showed us a 3 phased system that they engineered for their way of spam detection. In the first phase the user creates the rule for classification. Rules are nothing, but the keywords/phrases that occur in mails for respective legitimate or spam mails. The second phase can be called as training phase. Here the classifier will be trained using a spam and legitimate emails manually by the user. Then with the help of algorithm the keywords are extracted from classified mails. When the first and second phases are completed, classifying the emails by given algorithm starts, using this knowledge of tokens, the filter classifies every new incoming email. Here the probability of maximum keyword match is calculated and the status of a new email is confirmed as spam or important email (Figure 2.1).

When we researched about choosing a supervised machine learning algorithm we also encountered some issues that comes with supervised learning. The figure 2.2 explains about how a traditional supervised model works.

The first step is collecting the dataset. If a requisite expert is available, then s/he could suggest which fields (attributes, features) are the most informative. If not, then the simplest method is that of “brute-force,” which means measuring everything available in the hope that the right (informative, relevant) features can be isolated. However, a dataset collected by the “brute-force” method is not directly suitable for induction. It contains in most cases noise and missing feature

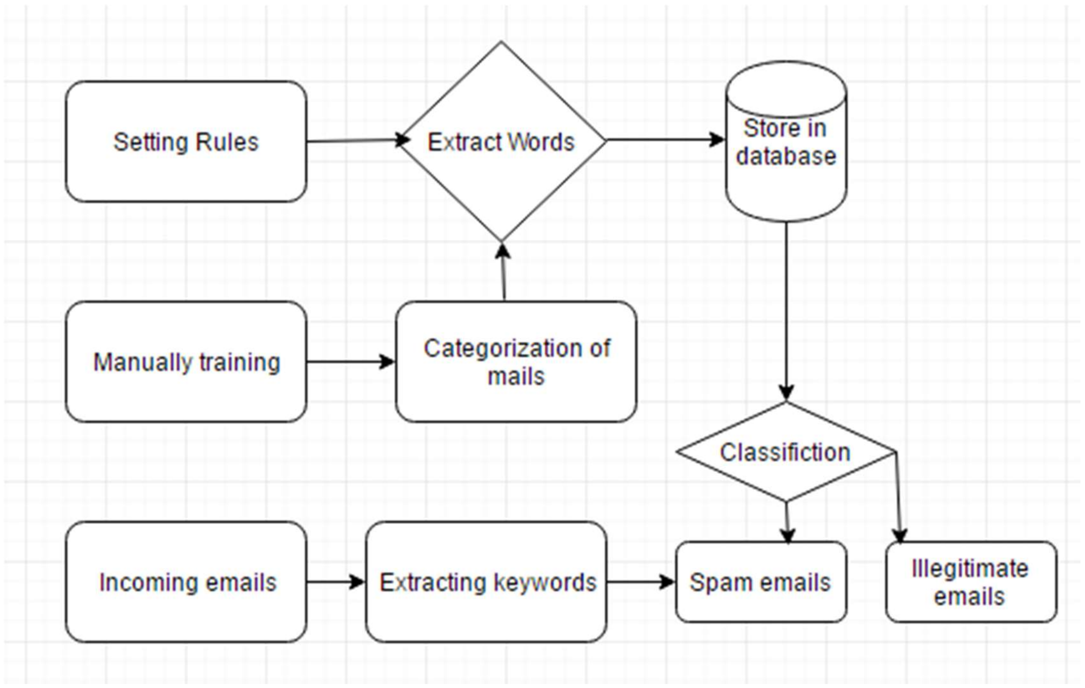


Fig 2.1: Implementation Diagram [7]

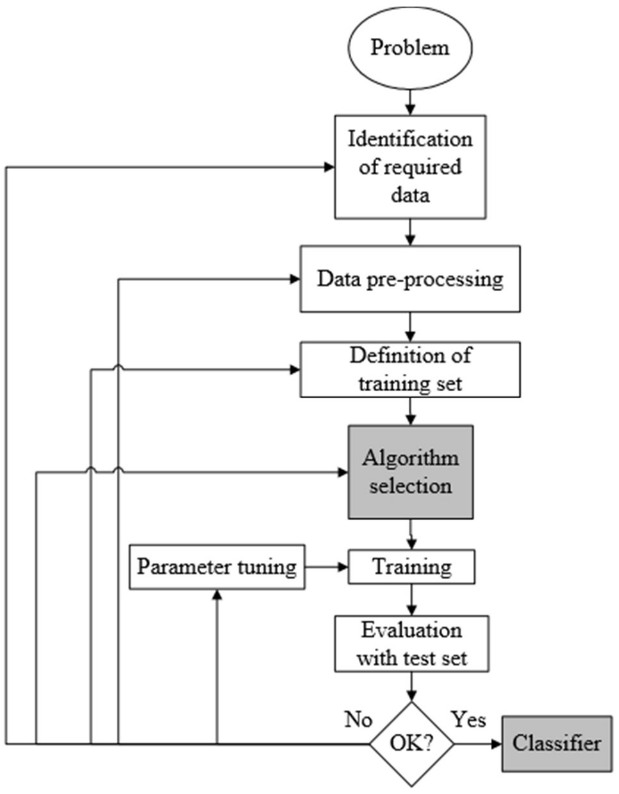


Fig 2.2: Workflow of a Supervised Machine Learning [8]

values, and therefore requires significant pre-processing S. B. Kotsiantis [8]

Jaswal, V. et al [9] worked on a spam detection system that uses detect spam words. They rely on filtering methods to detect stemming words of spam images and then use HMM of spam filters to detect all the spam images. They showed a methodology of spam detection in 4 steps,
Step 1 - to design a spam detection system.
Step 2 - to select a spam file either it is text file or it is excel file.
Step 3 - to select the file on the basis of spam detection.
Step 4 - filter stemming words only from spam detection

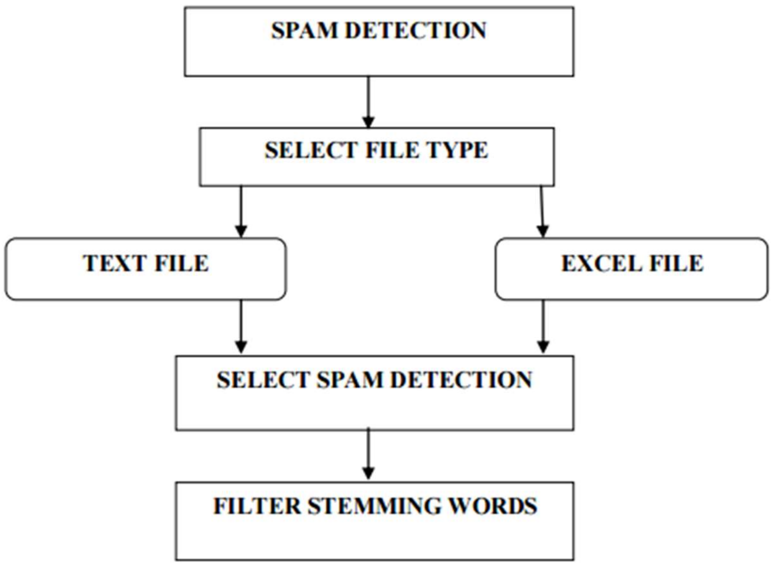


Fig 2.3: Workflow Model of Spam Detection [9]

Another work by Carpinter, J. et al [10] showed an approach in which their primarily focuses were on automated, non-interactive filters, with a broad review ranging from commercial implementations to ideas confined to current research papers. Both machine learning and non-machine learning based filters are reviewed as potential solutions and a taxonomy[7] of known

[7] taxonomy: a scheme of classification

approaches presented. They showed various approaches to spam filtering,

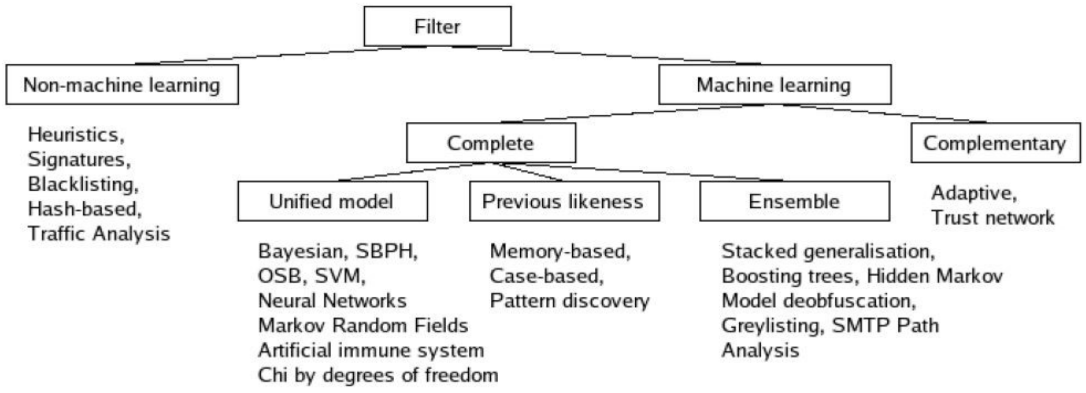


Fig 2.4: Classification of Filters [10]

Another study has been done by Kidmose, E. [11] where HMM was used to combined to estimate the life-cycle state of hosts, only relying on data observable in the network. We got some idea of transition probability from this paper. It shows for any state at any time the probability of arriving there only depends on the state distribution probability for the previous observation and the static transition probabilities.

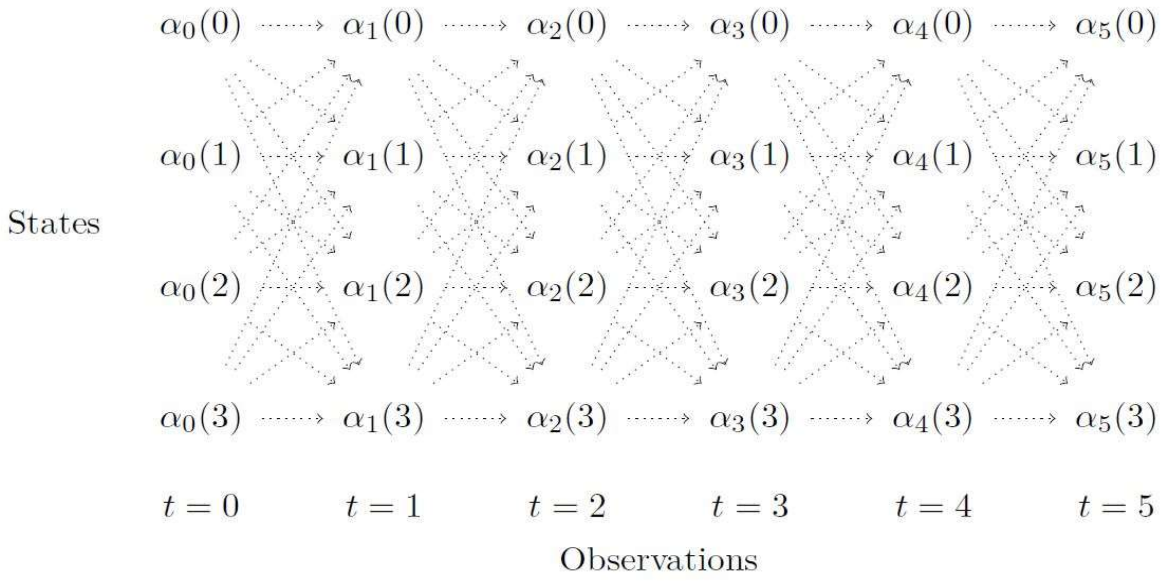

$\alpha_0(0)$ $\alpha_1(0)$ $\alpha_2(0)$ $\alpha_3(0)$ $\alpha_4(0)$ $\alpha_5(0)$
$\alpha_0(1)$ $\alpha_1(1)$ $\alpha_2(1)$ $\alpha_3(1)$ $\alpha_4(1)$ $\alpha_5(1)$
States
$\alpha_0(2)$ $\alpha_1(2)$ $\alpha_2(2)$ $\alpha_3(2)$ $\alpha_4(2)$ $\alpha_5(2)$
$\alpha_0(3)$ $\alpha_1(3)$ $\alpha_2(3)$ $\alpha_3(3)$ $\alpha_4(3)$ $\alpha_5(3)$
$t = 0$ $t = 1$ $t = 2$ $t = 3$ $t = 4$ $t = 5$
Observations

Fig 2.5: Transition Probability [11]

CHAPTER 3

Terminologies

To accomplish our desired result we had to use various kinds of algorithms, toolkits, libraries, datasets and classification methods. We have explained about all of these things under this chapter. Several NLTK toolkits were used for tokenizing and string manipulation. Powerful libraries such as stop words, lemmatization, stemming, POS tagging, chunking and chinking were used for fine tuning our working data. Vast online resources of NLTK toolkits made NLP very comfortable to us. Scikit-learn package for python is also described. This open source package with efficient tools- machine learning and data analysis helped us to implement HMM. The two algorithms that we used for our classification are discussed. The last segment of this chapter is dataset. As we used supervised learning method in our research we had to look for pre classified email dataset and we also analyzed real time email data for scheduling.

3.1 Natural Language Processing Tools

These tools below were mainly used for working with text type data. Most of these toolkits or libraries were used within python environment as they provide powerful online resources.

3.1.1 NLTK Toolkit

NLTK is one of the dominant platforms for creating Python programs that deals with word processing, tagging and string manipulations. It gives access to more than 50 corpora and lexical resources. There are also various text processing libraries for classification, tokenization, stemming, tagging, parsing, semantic reasoning and wrappers for industrial-strength NLP libraries. NLTK is intended to support research and teaching in Natural Language or closely related areas.

3.1.2 Stemming

Stemming is the technique of decreasing deviating or derived words to their base form. For grammatical reasons, documents are going to use different forms of a word, such as *meet*, *meets*, and *meeting*. In many situations, it is useful for a search for one of these words to return documents that contain another word in the set. Using stemming on the above strings, we will get *meet* as the base form. Stemming chops off the ends of words. Algorithms for stemming have been studied in computer science since the 1960s. The most common and effective algorithm for stemming English is *Porter's algorithm*. We have imported Porter Stemmer [12] from NLTK for stemming purpose.

3.1.3 Lemmatizing

Lemmatization is the process of converting the words of a sentence to its dictionary form. For example, given the words amusement, amusing, and amused, the lemma for each and all would be amuse. This aims to remove inflectional endings and to return base or dictionary form of a word. This process involves linguistic approach, such as morphological analysis through regular relations compiled in finite-state transducers. Importance of lemmatization is very high in this project. As to find out the exact meaning of a mail depends on the words. Those words are checked with some previous stored words to find whether it matches to a certain label. In this project the word “meeting” was one of those most checked words. Now this “Meeting” can come up with the same meaning by different form like “meet”, “met”, “meets”. But if we directly check all these word with one word “meeting” it will not get matched. So, to make a list of multiple words for only one meaning doesn’t come up with efficiency. Thus the necessity of the lemmatization comes in.

3.1.4 Stop Words

Stop words is a set of commonly used words in any language which are excluded out before or after processing of natural language data which, in our case, is text. The main reason why stop words are essential to any program is that, when we remove the words that are very commonly used in a given language, we can focus on the important words instead. For removing stop words from a document of our program we searched them in NLTK toolkit’s given list and the result we got was very accurate.

3.1.5 Chunking and Chinking

Chunking is an analysis of a sentence or phrase which first identifies constituent parts of sentences and then links them to higher order units that have discrete grammatical meanings (noun groups or phrases, verb groups, etc.). Some fundamental chunking algorithms simply join constituent unit on the basis of search patterns. On the other hand approaches that use machine learning techniques (classifiers, topic modeling, etc.) can analyze contextual information and thus create chunks in a way that they show the semantic relations between the basic constituents.
A Chunk Rule specifies what to include in a chunk, while a Chink Rule specifies what to exclude from a chunk. In other words, chunking creates chunks, while chinking breaks up those chunks. In our Scheduling algorithm we used elementary or fundamental chunking method.

3.2 Scikit-learn

Scikit-learn is an open source package of efficient tools for data mining, machine learning and data analysis. It’s a Python module that integrates a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. The Scikit-learn package emphasizes to bring machine learning to non-experts by using a general-purpose high-level language. Focuses on putting on ease of use, performance, documentation, and API consistency [19]. It is built on NumPy, SciPy and matplotlib. The Scikit-learn python module made it comfortable to implement the machine learning techniques and evaluate the findings and results of our experiments that was coded in Python programming language. We used Scikit-learn version 0.16.1 [21] to implement the HMM algorithm to fulfill our requirements and run the program successfully.

3.3 Classification

Text Classification issues single or multiple classes to a document according to their content and context. Classes are chosen from a previously fixed taxonomy (a hierarchy of categories or classes). Generally a text Classification algorithm requires some tasks (extracting text, tokenization, stop words removal and lemmatization). Our version of the algorithm combines statistical document classification with rule-based filtering, which allows to obtain a high degree of precision in a wide range of environments. Statistical classifiers provide a means to use example documents to define each category. In turn, rule base classifiers may help to fine-tune the classification and correct the output of statistical classifiers.

3.4 Naive Bayes Classification

Naive Bayes classifier is a machine learning algorithm. This Bayesian Classification[8] is used as a probabilistic learning method. It is not a single algorithm but a family of algorithms that all share a common principle, that every feature being classified is independent of the value of any other feature.

[8] See Appendix A.2

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. For example, a fruit may be considered to be an apple if it is red, round, and about three inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). [16] Let's look at the equation below:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \dots\dots\dots (3.1)$$

Above,

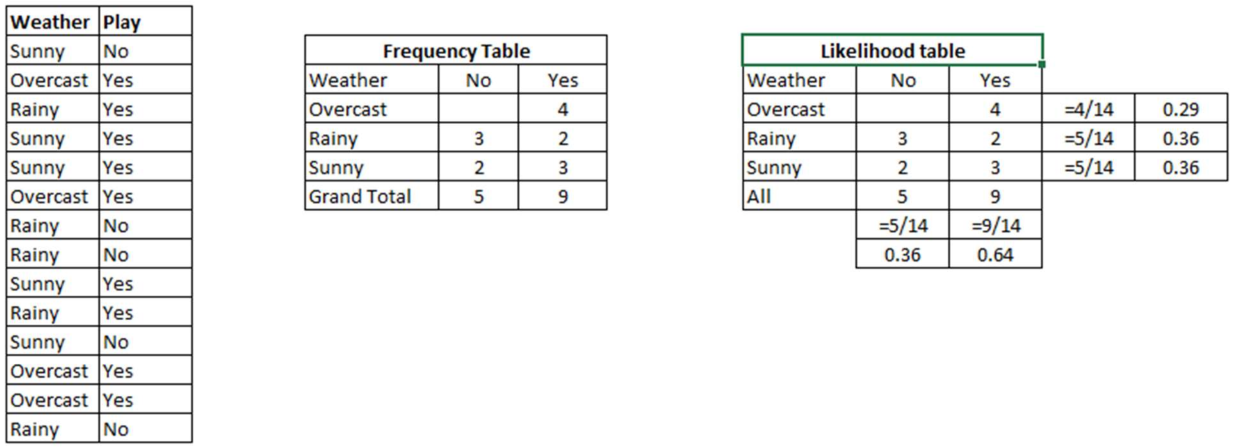
- *P*(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

3.4.1 How Naive Bayes algorithm works?

Let's understand it using an example. Below we have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.



Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Fig 3.1: Naive Bayes Example

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction. So, if there goes a problem like Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

$$P(Yes \mid Sunny) = P(Sunny \mid Yes) * P(Yes) / P(Sunny) \; \ldots \; \ldots \; \ldots \; (3.2)$$

P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64
Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability. Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Naive Bayes classifiers are among the most successful known algorithms for learning to classify text documents. That is why we chose to work with this.

3.4.2 Spam filtering

An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification. We have used 5500 emails from Enron email dataset. These emails have been stored separately according to their category. We have filtered the words $(w_1, \ldots, w_n)$ by removing stopwords and lemmatizing the words. For naive bayes classifier, we take the words $(w_1, \ldots, w_n)$ of an email and calculate the value of $P(w_1, \ldots, w_n)$

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i) \quad \ldots \ldots \ldots \; (3.3)$$

Now, let's denote an email as a capital letter *E*, two classes of emails as Important *(I)* and spam *(S)*. What we need to know next are the two probabilities *P(E|I) and P(E|S)* and we have to make an assumption of conditional independence. The assumption here is that the appearance of a particular token w is statically independent of the appearance of any other tokens $w_j, j \neq i$ given that we have either an important or a spam email. Then we can express *P(E|I) and P(E|S)* as:

$$\begin{aligned} P(E|I) &= P(w_1, \ldots, w_n|I) \\ &= \prod_{i=1}^{n} P(w_i|I) \quad \ldots \ldots \ldots (3.4) \end{aligned}$$

And

$$\begin{aligned} P(E|S) &= P(w_1, \ldots, w_n|S) \\ &= \prod_{i=1}^{n} P(w_i|S) \quad \ldots \ldots \ldots (3.5) \end{aligned}$$

The reason that we set up a training dataset is to estimate the nature of each word, where probabilities $P(w_i|I)$ *and* $P(w_i|S)$ are needed.

They denote the conditional probability that a given email contains the word w_i under the assumption that this email is spam or important respectively. We estimate these probabilities by calculating the frequencies of the words appear in either groups of emails from the training dataset. In the following formula, $P(w_i \;\cap\; S)$ is the probability that a given email is an important email and contains the word w_i . Thus, by Bayes theorem:

$$P(w_i|S) = \frac{P(w_i \ \cap \ \ S)}{P(S)} \quad \dots\dots\dots (3.6)$$

And

$$P(w_i|I) = \frac{P(w_i \ \cap \ \ I)}{P(I)} \quad \dots\dots\dots (3.7)$$

The following step is to compute the posterior probability of important email given the overall probability of the sampling email by Bayes' rule, this is the crucial part of the entire classification.

$$P(I|E) = \frac{P(E|I)P(I)}{P(E)} \quad \dots\dots\dots\dots (3.8)$$

$$= \frac{P(I)\prod_{i=1}^{n} \ P(w_i|I)}{P(E)} \quad \dots\dots\dots (3.9)$$

And similarly

$$P(S|E) = \frac{P(E|S)P(S)}{P(E)} \quad \dots\dots\dots\dots\dots\dots (3.10)$$

$$= \frac{P(S)\prod_{i=1}^{n} \ P(w_i|S)}{P(E)} \quad \dots\dots\dots (3.11)$$

Therefore we can classify the email by comparing the probabilities of *P(I|E) and P(S|E)*. Firstly, we find the ratio of these two probabilities. In the equations below, the denominators *P(E)* from (3.8) and (3.10) cancel out each other.

$$\frac{P(S/E)}{P(I/E)} = \frac{P(E|S)P(S)}{P(E|I)P(I)} \quad \dots\dots\dots (3.12)$$

$$= \frac{P(S)\prod_{i=1}^{n} \ P(w_i|S)}{P(I)\prod_{i=1}^{n} \ P(w_i|I)} \quad \dots\dots\dots (3.13)$$

$$= \frac{P(S)}{P(I)} \prod_{i=1}^{n} \frac{P(w_i|S)}{P(w_i|I)} \quad \ldots\ldots\ldots (3.14)$$

But there is a problem here, the products in the above equations can be extremely small values if we have a big amount of words w_i. To overcome this issue, we apply log to the probability ratio.

$$log \frac{P(S|E)}{P(I|E)} = \log\left(\frac{P(S)}{P(I)} \prod_{i=1}^{n} \frac{P(w_i|S)}{P(w_i|I)}\right) \quad \ldots\ldots.\ldots (3.15)$$

$$= log \frac{P(S)}{P(I)} + \sum_{i=1}^{n} log \frac{P(w_i|S)}{P(w_i|I)} \quad \ldots\ldots\ldots \ (3.16)$$

At this point, we can use equation (3.16) to calculate the log posterior probability when we receive a new email. If the result is greater than zero (which means *P(S|E) > P(I|E))*, we classify email E as spam. Similarly, we classify the email as important if it is less than zero (which means *P(S|E) < P(I|E))*.

3.5 Hidden Markov Model (HMM)

HMM is a tool for representing probability distributions over sequence of observations. The HMM assumes that the observation at time *t* was generated by some process whose state *St* is hidden from the observer. It also assumes that the state of this hidden process satisfies the markov property, which is, given the value of St-1, the current state St is independent of all the states prior to t-1. [17] Graphically we can explain it as shown in figure 3.1.

The graph shows the dependencies between the variable of the model. S = {S_1, S_2, S_3,....S_t} is a sequence of states, we do not observe S. Y = {Y_1,Y_2,Y_3,...Y_t} is a sequence of emissions, we

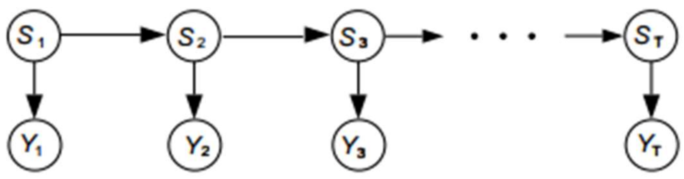


Fig 3.1: Trellis Diagram

observe Y. Y_2 is conditionally independent of everything else given S_2. S_4 is conditionally independent of everything else given S_3. Probability of being in a particular state at step *i* is known once we know what state we were in at step *i-1*. Probability of seeing a particular emission at step *i* is known once we know what state we were in at step *i*. The joint distribution of a sequence of states and observations can be factored in the following way: [20]

$$P(S_{1:T}, Y_{1:T}) = P(S_1)P(Y_1|S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})P(Y_t|S_t) \quad \ldots\ldots\ldots (3.17)$$

To describe HMM even further, we will use an example that is easy to understand. Let's say father brings 3 types of snacks at home: hotchpotch, noodles, and ice-cream. And let us suppose that each day of the week may be classified as either of the following: rainy, cold, or hot. But we have no way of ascertaining what sort of day it is.

We want to develop a model that predicts which snacks father is going to bring home on a particular day, *if we know the sequence of snacks that he has been bringing home for the past month*. The observations (father's snacks) are dependent on some process which is *hidden* from our view (the weather).

We assume that each day's weather is dependent on and only on the previous day's weather. This way the sequence of weather description turns out to be a Markov chain. The probability of today's weather, given that we know yesterday's weather forms the 'transition probability' of the Markov Chain. For example, if yesterday was rainy, the probabilities of today's weather may be 0.5 for rainy, 0.3 for cold and 0.2 for hot. The matrix formed by taking all such probabilities is called the 'Transition probability matrix'.

Now, we know that father's snacks depends on each day's weather. Like, if it is rainy, father is more likely to bring home some hotchpotch. Hence the probabilities may be 0.7 for hotchpotch, 0.2 for noodles and 0.1 for ice-cream. We construct a matrix containing all such probabilities for all weather conditions. This matrix is called the 'Emission probability matrix'.

To fully populate the HMM, we need:

1. Start probabilities – What are the chances of starting in a state? Or in other words – what weather is it more likely to be today?

In equation (3.17),

$$P(S_1)P(Y_1|S_1)\ is\ the\ start\_probability.$$

2. Emission probabilities – What are the chances of each observation occurring in a state? What are the chances of father bringing hotchpotch, noodles or ice-cream on a rainy day?

In equation (3.17),

$$P(Y_t|S_t)\ is\ the\ emission\_probability$$

3. State Transition probabilities – How frequently do the states change? What are the chances of today being rainy, cold or hot given yesterday was rainy?

In equation (3.17),

$$P(S_t|S_{t-1})\ is\ the\ transitio\ \ \_probability$$

3.6 Dataset

Our dataset contains three types of data they are pre classified (already sorted each document under their designated title), Real time emails and data from Google calendar entries. Pre classified data is used in our system to train our learning algorithm and we tested them with real time incoming emails. This emails were further analyzed with scheduling algorithm matching them with data from Google calendar event which were fetched by Google calendar API.

3.6.1 Pre-classified Data

We have looked into many online resource for pre-classified email dataset. Then we have used 5500 emails from Enron Email Dataset. [22] 1500 of them are important emails and we have stored them in a folder titled "important". The other 4000 emails are spam emails which we have stored in a "spam" folder. Spam emails are greater in number because we have noticed that negative recall increases substantially when we use more spam emails in our dataset.

3.6.2 Real-time Emails

We have used python's IMAP and http library to log into a gmail account and we fetched unseen emails for processing. Each email is striped off HTML and words are tokenized and then pos tagged. Then we have eliminated unwanted words and analyzed for any meeting related info. For this purpose we prepared some regular expression rules and applied chunking and chinking techniques to identify if the document is meeting related. After identifying a meeting email we extracted time and date related information in the basis of pos tagging.

3.6.3 Google Calendar API

Another sort of data we have used in our project is Calendar events and we have used

Google Calendar API for this. From Google calendar we have extracted events of a user showing all of their meetings, schedules and events. This extracted information is used to identify free slots and occupied slots. After extracting times and dates from emails (which we talked in 4.2) we then compare those with Google calendar events. If that time is already booked or excluded for any purpose by the user, no meeting gets fixed. Otherwise a meeting is automatically scheduled.

3.7 System Setup

Hardware and software used in this research played a big role in terms of results. Both hardware and software specifications have been mentioned here.

3.7.1 Hardware Specification

CPU:

Name	AMD FX(tm)-8300
Cores	8
Clock speed (mhz)	3300
Typical TDP	95W
Socket	Socket AM3+
Microarchitecture	Piledriver
Platform	Volan
Processor core	Vishera
Core stepping	OR-C0
CPUID	600F20

Manufacturing process	0.032 micron
Data width	64 bit
Level 1 cache size ?	4 x 64 KB 2-way set associative shared instruction caches 8 x 16 KB 4-way set associative data caches
Level 2 cache size ?	4 x 2 MB 16-way set associative shared exclusive caches
Level 3 cache size	8 MB 64-way set associative shared cache

Table 3.1: CPU Specification

Memory:

Physical memory	16GB

Table 3.2: RAM

GPU:

GPU	NVIDIA GeForce GT 620

Table 3.3: GPU Specification

3.7.2 Software Specification

Name	**Type**	**Version**	**Architecture**
Anaconda	Python distributer	Anaconda2 4.2.0 Python 2.7.12	32 bit (x86)
Pycharm	Text Editor	2016.2.3 Build #PC 162.1967.10.	32 bit (x86)
SciKit Learn	Python package	0.16.1	32 bit (x86)
NLTK	Natural Language Toolkit	3.2.1	32 bit (x86)

Table 3.4: Software Requirements

OS:

Name	Microsoft Windows 10 Pro
Version	10.0.10586
Build Number	10586
System type	64 bit

Table 3.5: Operating System Details

CHAPTER 4

System Implementation

In the previous chapter, even though we have discussed most of the terminologies as well as the algorithms- Naïve Bayes and HMM, we made a lot of changes to both the algorithms for them to work accordingly in our system setup. In this chapter, we have talked about those changes, our own algorithms and demonstrated our working procedure. First, we have described how we used Naïve Bayes and HMM for email classification. Then, we have mentioned the details of our meeting scheduling algorithm.

4.1 Email Classification

We stored 1500 important emails and 4000 spam emails into a python dictionary and named it 'documents'. Some pre-processing needed to be done before we ran classifier algorithms on these emails. For every email, we ran some methods on that email until we reached the end of the dictionary. At first, we imported *message_from_string* from *email* module and then we called *get_payload* method on it to get rid of the email multipart problem. This method returns the current payload, which will be a list of Message objects when *is_multipart()* is True, or a string when *is_multipart()* is False. We also imported *BeautifulSoup* from *bs4* module and used it to strip HTML off the email. Then, we got the email texts only. We imported *split* from *re* module and applied the method on email text, which left us with the tokenized words from these 5500 emails. At this point, we have tried various combinations of NLP techniques- stemming, lemmatizing and stopwords removing. We looked for a single technique or a combination of these technique that would give us the best result. We used frequency distributor to order the words based on the number of times they appear in the pre-classified emails. Two different lists were used to store the most frequent words from important emails and spam emails. Then we took 2650 words from each of these lists (to limit the number of features that the classifier needs to process) and shuffled them into a list called *word_features*.

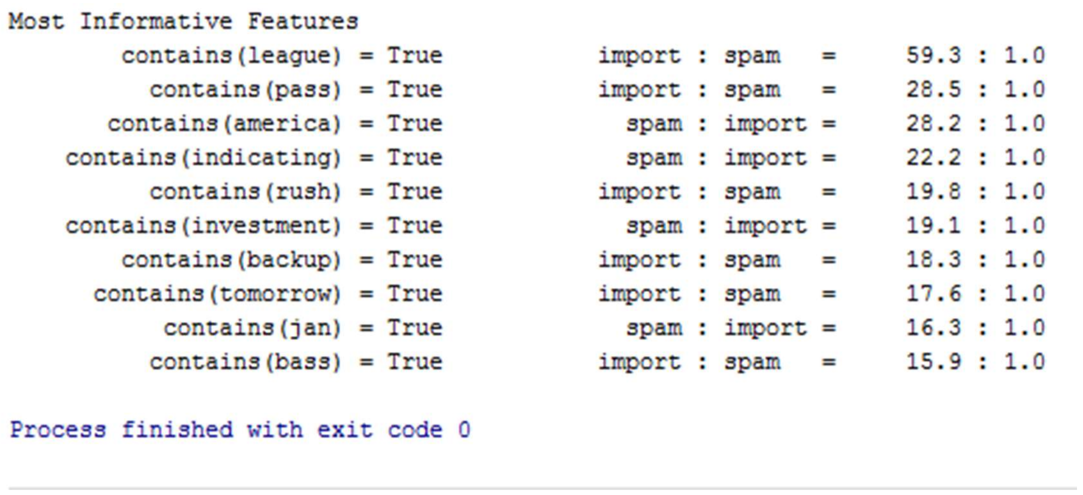
4.1.1 Naive Bayes Classifier

The goal is to build a classifier that will automatically tag new emails with appropriate category labels. We have a list of documents at hand- emails labeled with the appropriate categories. The first step in creating a classifier is deciding what features of the input are relevant, and how to encode those features. [18] So, we define a feature extractor for documents so that the classifier knows which aspects of the data it should pay attention to. We took into account each email from *'documents'* and used the *set* method to remove the duplicate words from those emails. This makes the checking faster. Now for every word in *word_features*, if that word existed in a given email, we associated the word with the category (important or spam) of that email. Thus, we found words that were labeled as 'important' and words labeled as 'spam'. And these *words:label* pairs were used as *featureset*[9] for Naive Bayes Classifier. We realize that there are words in featureset that are labeled as both important and spam. Now that we've defined our feature extractor, we can use it to train a classifier to label new emails. We used ninety percent of the featureset as *train_set* while the remaining ten percent was used as *test_set*. To check how reliable the resulting classifier is, we compute its accuracy on the test_set. And we can use

[9] featureset: A dictionary, maps from feature names to their values

show_most_informative_features() to find out which features the classifier found to be most informative.

Apparently in this email dataset, an email that mentions "investment"[10] is almost 19.1 times more likely to be spam than important, while an email that mentions "appointment" is about 6 times more likely to be important.



```
Most Informative Features
        contains(league) = True           import : spam   =     59.3 : 1.0
          contains(pass) = True           import : spam   =     28.5 : 1.0
       contains(america) = True             spam : import =     28.2 : 1.0
    contains(indicating) = True             spam : import =     22.2 : 1.0
          contains(rush) = True           import : spam   =     19.8 : 1.0
    contains(investment) = True             spam : import =     19.1 : 1.0
        contains(backup) = True           import : spam   =     18.3 : 1.0
      contains(tomorrow) = True           import : spam   =     17.6 : 1.0
           contains(jan) = True             spam : import =     16.3 : 1.0
          contains(bass) = True           import : spam   =     15.9 : 1.0

Process finished with exit code 0
```

Fig 4.1: Most informative features (Naive Bayes)

Therefore, even though most words could be labeled as both important and spam in the early stage, *show_most_informative_features()* method gives us a ratio which helps to determine the ultimate label of any such word. If the ratio is 1, only in that case a word is considered to be both important and spam.

We also tweaked *show_most_informative_features_in_list(classifier, n)* method to get a *word:label* pair where the most informative features were labeled as important or spam. We extracted the spam words and stored them in a list called *spamwords*, similarly we stored the important words in a list called *importantwords*.

[10] See figure 4.1

Then we took into consideration the 200 pre-classified emails we used for testing- 100 important emails and 100 spam emails. We split the emails into words and checked how many of those words existed in *importantwords* and how many of them existed in *spamwords.* If there were more *importantwords,* the email was classified as an *important* email. If *importantwords* and *spamwords* were equal, we could not determine the email category. Otherwise, the email was classified as a *spam* email. Finally, we calculated the accuracy based on the number of important emails and spam emails that were classified correctly by our algorithm. Then we tried various combinations of NLP techniques- stopwords removing, stemming, lemmatizing on Naïve Bayes Classifier to inspect the differences in accuracy as well as to find the best method among them. The variants were: 1. Basic Naive Bayes, 2. Removing Stopwords, 3.Stemming, 4. Lemmatizing, 5. 'Removing Stopwords' and Stemming, 6. 'Removing Stopwords' and Lemmatizing, 7. Lemmatizing and Stemming, 8. 'Removing Stopwords', Lemmatizing and Stemming. Among all these combinations, 'removing stopwords' and lemmatizing gave the most accuracy.

4.1.2 HMM

The goal is same here- to build a classifier that will automatically tag new emails with appropriate category labels. We used two states- *important* and *spam*. *Word_features* were used as observations. We used *start_probability* {'important': 0.5, 'spam': 0.5}. Then we created a matrix and named it *y* which looks like the following:

word_features	spam	important
Meeting	4	20
Discount	18	3
Assignment	2	12
Office	6	23
Coupon	11	0
Offer	13	6
Valid	10	8

Table 4.1: Hidden Markov Model (word_features)

For each word in word features (observations), it counts the number of times that word appears in important emails and spam emails. We used the following algorithm to populate this matrix:

```
for each word in observations:
    for each email in documents:
        for each word in email:
            if the word matches with an observation:
                if the email is spam:
                    increment y[word][spam]
                else:
                    increment y[word][important]
```

The *emission_probability* represents how likely a word is to be an important word or a spam word for each state. For instance, the word ‘discount’ has a 10% chance to appear in an important email while the same word has a 60% chance to appear in a spam email. We have used the following algorithm to get the emission matrix:

```
for each pair in y:
        for each element in the pair:
                append the first element to emission_probability[spam]
                increment spamcount
                append the second element to emission_probability[important]
                increment importantcount

for each state in emission_probability:
        for each observation in the state:
                if state is spam:
                        emission_probability[spam][observation]                    =
                        emission_probability[spam][observation]/ spamcount
                else:
                        emission_probability[important][observation]               =
                        emission_probability[important][observation]/ importantcount
```

The *transition_probability* represents the change of the state in the underlying Markov chain. For instance, if a word is spam, transition_probability dictates how likely the next word is going to be spam as well. We have used the following algorithm:

```
for each pair in y:
   if y[word][spam] is greater than y[word][important]:
   //first word is spam
   //check for next word
      if y[next word][spam] is higher than y[next word][important]:
          //immediate word after spam is also a spam
          increment transition_probability [spam][spam]
      else if y[next word][spam] equals y[next word][important]:
          //immediate word after spam has equal chance of being a spam or important word
          increment transition_probability[spam][spam]
          increment transition_probability[spam][important]
      else:
          //immediate word after spam is an important word
          increment transition_probability[spam][important]
   else if y[word][spam] is less than y[word][important]:
      //first word is important
      //check for next word
      if y[next word][spam] is greater than y[next word][important]:
          //immediate word after important is a spam
          increment transition_probability [important][spam]
      else if y[next word][spam] equals y[next word][important]:
          //immediate word after important has equal chance of being a spam or important word
          increment transition_probability[important][spam]
          increment transition_probability[important][important]
      else:
          //immediate word after important is an important word
          increment transition_probability[important][important]
//check next word
```

Total Words After Spam is the summation of transition_probability[spam][spam] and transition_probability[spam][important]. We get the new transition_probability[spam][spam] by dividing the old transition_probability[spam][spam] by Total Words After Spam. Similarly we get transition_probability[important][important]

We used *hmm* from *sklearn* module and set the start_probability, transition_probability and emission_probability. Then, like in naive Bayes classifier, we took 200 pre-classified emails for testing- 100 important emails and 100 spam emails. We split the email into words and added those words as observations and named the list *observe*. If there were more *importantwords,* the email was classified as an *important* email. If *importantwords* and *spamwords* were equal, we could not determine the email category. Otherwise, the email was classified as a *spam* email. HMM algorithm was also run with eight different combinations of NLP techniques like Naive Bayes algorithm. This time we got a different result. HMM with stemming outperformed all the other seven variants. After that we compared the best result of Naive Bayes algorithm with the best result of HMM. Overall HMM was more successful in identifying spam and important emails from the test dataset. We later chose HMM to be our primary classifier for the system.

4.2 Meeting Scheduling

After getting a new mail, our meeting scheduling works in 3 basic process.
Step 1: Fetch unseen mail from server
Step 2: Determine whether the mail is important or spam
Step 3: Check if email contains meeting request

```
if (important)
    check if asks for meeting
    if (true)
        check time and date with calendar
        If clashes
            no meeting fixed
        Else
            Meeting fixed
```

Step 1. Fetching Mails from Server

First, our application gets connected over an SSL encrypted socket (to use this it needs a socket module that was compiled with SSL support). If *host* is not specified, '' (the local host) is used. If *port* is omitted, the standard IMAP4-over-SSL port (993) is used. *keyfile* and *certfile* are also optional - they can contain a PEM formatted private key and certificate chain file for the SSL connection. It needs imaplib library to get all these things done successfully. [23]

After getting connected successfully our application logs in to the user's account by another library called *getpass*. With the name it may seem that it only prompt the user for a password but this module comes up with two functions, *getpass.getpass*and *getpass.getuser*. The user is prompted using the string prompt with the first one which defaults to 'Password: ' and the second checks the environment variables *LOGNAME, USER, LNAME* and *USERNAME*, in order, and returns the value of the first one which is set to a non-empty string. If none are set, the login name from the password database is returned on systems which support the *pwd* module, otherwise, an exception is raised.

When gets logged in, it specifically goes to inbox. Then it searches for "unseen" message. If any new message arrives to inbox it gets automatically set to unseen. We then fetched the email and save it to a variable named "*raw_email*". You can simply guess with the name of the variable that it stores the raw email in the variable. Therefore we need to exclude those unnecessary information (widely known as multipart problem) to get the actual data of the email which will be used to determine whether it is an important or spam.

Therefore we used payload[11] to extract the original message. This process gives us only information of sender, recipient, subject and body. Then it Step 2 is executed.

[11] The term 'payload' is used to distinguish between the 'interesting' information in a chunk of data or similar, and the overhead to support it. This being carried within a packet or other transmission unit. The payload does not include the "overhead" data required to get the packet to its destination. To a communications layer that needs some

Step 2. Determine whether the mail is important or spam

The extracted body then passes through our spam detection algorithm (HMM) to find out if the mail was spam or important. After detecting the email with respective category, this email is saved to a folder with same category.

Step 3. Check if email contains a meeting request

For this project we took “meet” as the base word. Therefore, a list is taken to store all the synset[12] of the word “meet”. There are two more lists named dayNN and monthNN storing the names of all days and months respectively. We then created a regular expression which detects if that email falls under the meeting category. This expression is

PRP+VB || [meet] + CD

Any word could be chosen other than “meet” based on one’s preference and their needs and that’s what makes our approach versatile. In our project we have shown one example of classifying meeting related important emails. Same process can also be applied to determine if the email is about “Birthday” wishes or “Press Conference”.

of the overhead data to do its job, the payload is sometimes considered to include the part of the overhead data that this layer handles.

[12] A set of one or more synonyms that are interchangeable in some context without changing the truth value of the proposition in which they are embedded.

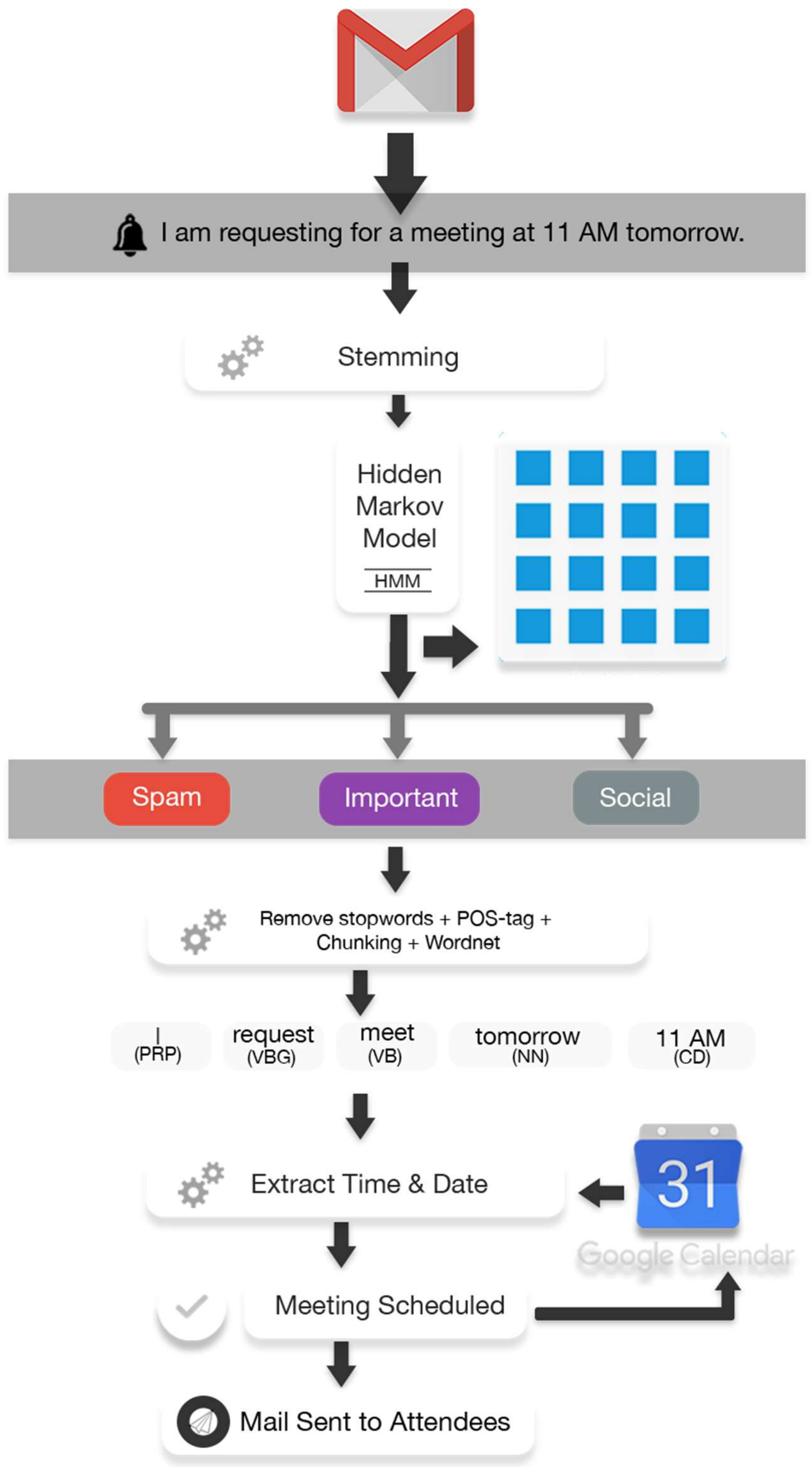


Fig 4.2: Work Flow

CHAPTER 5

Experiments and Result Analysis

In the previous chapter we have discussed about the system implementation of our research. We have demonstrated how we built our algorithms. Now we are going to talk about the results we obtained from our experiments upon the implementation of this system. This chapter describes the experiments in two parts. In the first part, we discuss about the classification results and analysis. Later we show the output generated by our intelligent email assistant.

5.1 Classification

The Results were found by experimenting on two machine learning algorithms with various combinations of different approaches and the result analysis are based on their accuracy, precision, recall and F-measure. [24]

Accuracy is the representation of how many true values are found comparing to all data. But it is not the only metric for evaluating the effectiveness of a classifier. Two other useful metrics are precision and recall.

Precision measures the exactness of a classifier. A higher precision means less false positives, while a lower precision means more false positives. This is often at odds with recall, as an easy way to improve precision is to decrease recall.

Recall measures the completeness, or sensitivity, of a classifier. Higher recall means less false negatives, while lower recall means more false negatives. Improving recall can often decrease precision because it gets increasingly harder to be precise as the sample space increases.

Accuracy is calculated using the formula, $\frac{TP+TN}{TP+TN+FP+F}$ - - - - - - - - - - - - - (5.1)

Positive Precision is calculated using the formula, $\frac{TP}{TP+FP}$- - - - - - - - - - - - -(5.2)

Negative Precision is calculated using the formula, $\frac{FN}{FN+TN}$- - - - - - - - - - - - -(5.3)

Positive Recall is calculated using the formula, $\frac{TP}{TP+FN}$- - - - - - - - - - - - - ----(5.4)

Negative Recall is calculated using the formula, $\frac{FN}{FN+TP}$- - - - - - - - - - - ----(5.5)

Where TP stands for true positive (actual and test data is accurately classified), TN stands for true negative (actual and predicted test data both are inaccurately classified), FP stands for false positive (actual data is inaccurate but predicted test data as accurate) and FN stands for false negative (actual data is accurate but predicted test data as inaccurate).

F-measure is used to measure an experiment's accuracy. F-measure can be interpreted as a weighted average of the precision and recall. F-score reaches best at 1 and worst at 0.

Positive F-measure formula, $2\cdot\frac{\text{positive } precision * \text{positive } recall}{\text{positive } precision + \text{positive } recall}$- - - - - - - - - - - ------(5.6)

Negative F-measure formula, $2\cdot\frac{\text{negative } precision * \text{negative } recall}{\text{negative } precision + \text{negative } recall}$- - - - - - - - - ------(5.7)

5.1.1 Classification Analysis

In total, there are 5500 email samples containing 1500 important mails and 4000 spam mails for training set and 200 emails containing 100 important mails and 500 spams for test set were taken for classification analysis.

5.1.1.1 Experiment with Naive Bayes

As stated earlier classifying with Naive Bayes algorithm using basic approach along with various combinations of 3 different processes; Stop Words, Stemming and Lemmatization were used in this experiment. Table 5.1 describes evaluation on test set using basic approach along with various combinations of 3 different processes in Naive Bayes algorithm. Figure 5.1 shows the comparison among different approaches used to classify emails based on Naive Bayes Algorithm. Figure 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9 were taken from output window of PyCharm which we used to run our code written in python.

In table 5.1, in total 200 emails were experimented as test set where 100 was important email and 100 was spam email. Some instances were classified correctly, some were classified incorrectly and very few emails remained indeterminate as the possibility of being both important and spam were 50% in these emails. The indeterminate instances were ignored in this case as they could be either important or spam. Using basic Naive Bayes approach out of 100 important emails 68 instances were classified correctly, 23 instances were classified incorrectly and 9 instances could not be determined. And out of 100 spams 69 instances were classified correctly, 15 instances were classified incorrectly and 16 instances could not be determined. The total accuracy achieved was 78.28%.

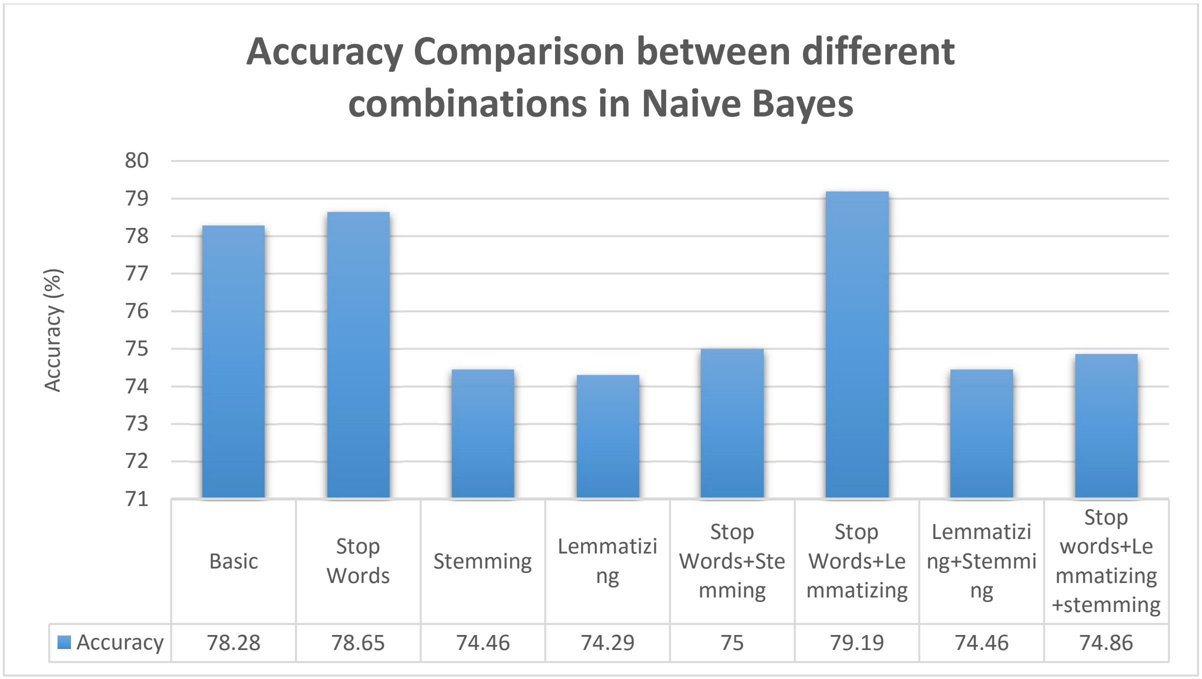
Then again using only stop words, out of 100 important emails 83 instances were classified correctly, 13 instances were classified incorrectly and 4 instances could not be determined. And out of 100 spams 57 instances were classified correctly, 25 instances were classified incorrectly and 18 instances could not be determined. The total accuracy achieved was 78.65%.

Using only stemming out of 100 important emails 85 instances were classified correctly, 7 instances were classified incorrectly and 8 instances could not be determined. And out of 100 spams 52 instances were classified correctly, 40 instances were classified incorrectly and 8 instances could not be determined. The total accuracy achieved was 74.46%. Using Lemmatizing out of 100 important emails 72 instances were classified correctly, 16 instances were classified incorrectly and 12 instances could not and out of 100 spams 58 instances were classified correctly, 29 instances were classified incorrectly and 13 instances could not be determined. The total accuracy achieved was 74.29%.

Process	**Trained data**	**Test data**	**Correctly Classified instances**		**Incorrectly Classified instances**		**Indeter- -minate instances**		**Accuracy*[13] (Mail Classifi-- cation)**
			Impor- -tant	Spam	Impor- -tant	Spam	Impor- -tant	Spam	
Basic	5500	200	68	69	23	15	9	16	78.28
Stop Words	5500	200	83	57	13	25	4	18	78.65
Stemming	5500	200	85	52	7	40	8	8	74.46
Lemmatizing	5500	200	72	58	16	29	12	13	74.29
Stop Words + Stemming	5500	200	74	58	14	30	12	12	75.00
Stop Words + Lemmatizing	5500	200	70	67	16	20	14	13	79.19
Lemmatizing + Stemming	5500	200	85	52	6	41	9	7	74.46
Stop Words + Lemmatizing + Stemming	5500	200	72	59	15	29	13	12	74.86

Table 5.1: Evaluation on Test Set (Naive Bayes in Different Processes)

[13] The accuracy was calculated considering only the instances that could be determined.



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing +stemming
■ Accuracy	78.28	78.65	74.46	74.29	75	79.19	74.46	74.86

Fig 5.1: Accuracy Comparisons (Naive Bayes)

Using both stop words and stemming out of 100 important emails 74 instances were classified correctly, 14 instances were classified incorrectly and 12 instances could not be determined. And out of 100 spams 58 instances were classified correctly, 30 instances were classified incorrectly and 12 instances could not be determined. The total accuracy achieved was 75%.

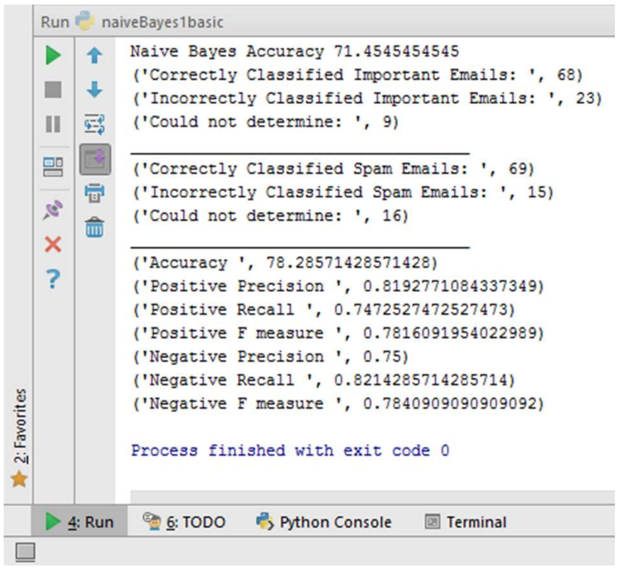
Using both stop words and lemmatizing out of 100 important emails 70 instances were classified correctly, 16 instances were classified incorrectly and 14 instances could not be determined. And out of 100 spams 67 instances were classified correctly, 20 instances were classified incorrectly and 13 instances could not be determined. The total accuracy achieved was 79.19%.

Using first lemmatizing then stemming out of 100 important emails 85 instances were classified correctly, 6 instances were classified incorrectly and 9 instances could not be determined. And out of 100 spams 52 instances were classified correctly, 41 instances were classified incorrectly and 7 instances could not be determined. The total accuracy achieved was 74.46%.

Lastly using stop words, lemmatizing and stemming out of 100 important emails 72 instances were classified correctly, 15 instances were classified incorrectly and 13 instances could not be determined. And out of 100 spams 59 instances were classified correctly, 29 instances were classified incorrectly and 12 instances could not be determined. The total accuracy achieved was 74.86%.

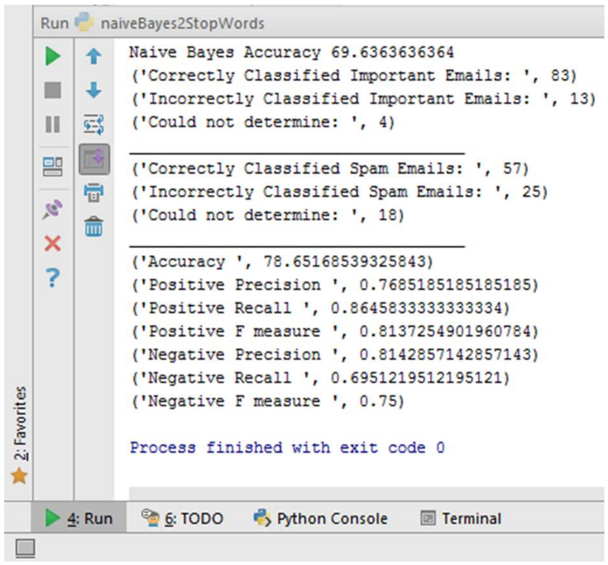
Figure 7.1 shows accuracy comparison among 8 different combinational approach to Naive Bayes and it shows that using stop words and lemmatizing together gives the best accuracy result which is 79.19%.

Detailed results found from different approaches using NLTK in python are shown below-



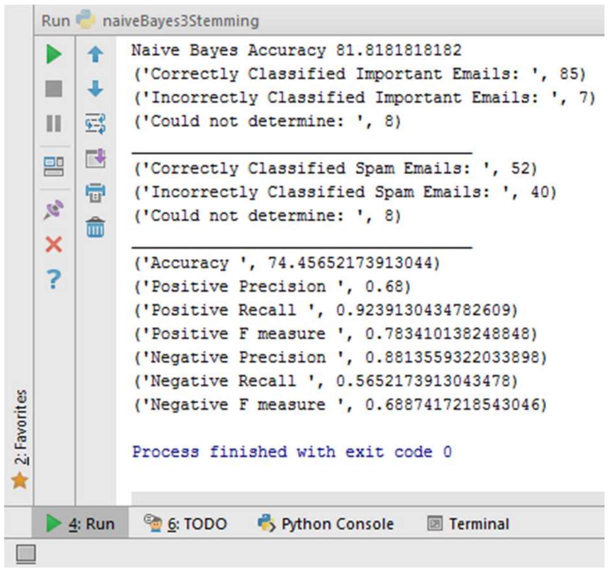
```
Naive Bayes Accuracy 71.4545454545
('Correctly Classified Important Emails: ', 68)
('Incorrectly Classified Important Emails: ', 23)
('Could not determine: ', 9)
_______________________________________
('Correctly Classified Spam Emails: ', 69)
('Incorrectly Classified Spam Emails: ', 15)
('Could not determine: ', 16)
_______________________________________
('Accuracy ', 78.28571428571428)
('Positive Precision ', 0.8192771084337349)
('Positive Recall ', 0.7472527472527473)
('Positive F measure ', 0.7816091954022989)
('Negative Precision ', 0.75)
('Negative Recall ', 0.8214285714285714)
('Negative F measure ', 0.7840909090909092)

Process finished with exit code 0
```

Fig 5.2: Results using Basic approach on Naive Bayes



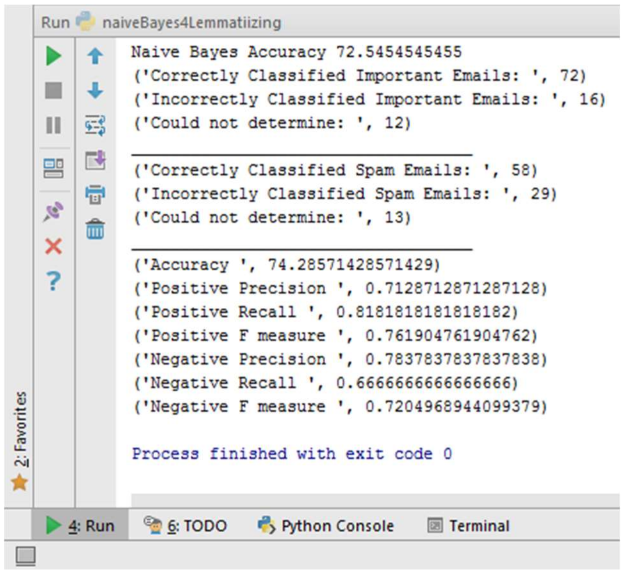
```
Naive Bayes Accuracy 69.6363636364
('Correctly Classified Important Emails: ', 83)
('Incorrectly Classified Important Emails: ', 13)
('Could not determine: ', 4)
_______________________________________
('Correctly Classified Spam Emails: ', 57)
('Incorrectly Classified Spam Emails: ', 25)
('Could not determine: ', 18)
_______________________________________
('Accuracy ', 78.65168539325843)
('Positive Precision ', 0.7685185185185185)
('Positive Recall ', 0.8645833333333334)
('Positive F measure ', 0.8137254901960784)
('Negative Precision ', 0.8142857142857143)
('Negative Recall ', 0.6951219512195121)
('Negative F measure ', 0.75)

Process finished with exit code 0
```

Fig 5.3: Results using Stop Words on Naive Bayes



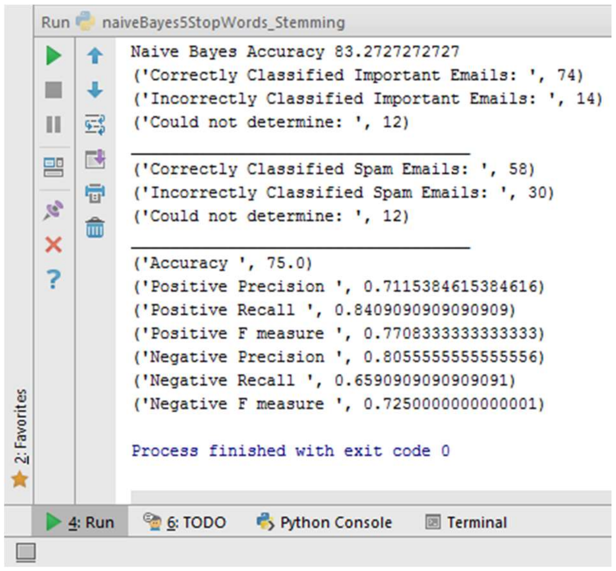
```
Naive Bayes Accuracy 81.8181818182
('Correctly Classified Important Emails: ', 85)
('Incorrectly Classified Important Emails: ', 7)
('Could not determine: ', 8)
________________________________________
('Correctly Classified Spam Emails: ', 52)
('Incorrectly Classified Spam Emails: ', 40)
('Could not determine: ', 8)
________________________________________
('Accuracy ', 74.45652173913044)
('Positive Precision ', 0.68)
('Positive Recall ', 0.9239130434782609)
('Positive F measure ', 0.783410138248848)
('Negative Precision ', 0.8813559322033898)
('Negative Recall ', 0.5652173913043478)
('Negative F measure ', 0.6887417218543046)

Process finished with exit code 0
```

Fig 5.4: Results using Stemming on Naive Bayes



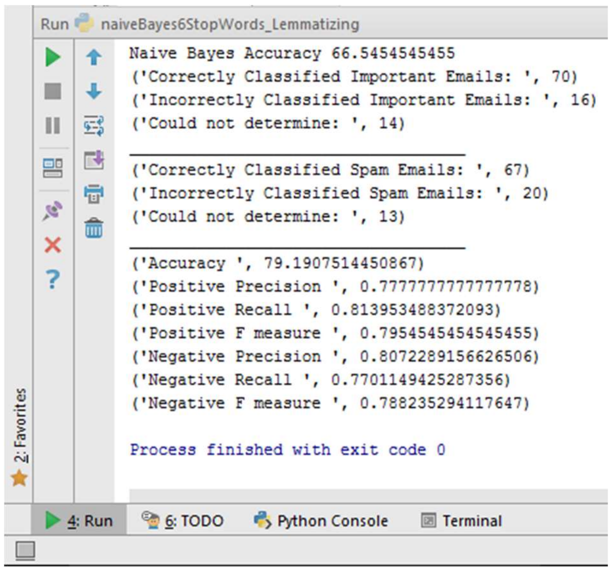
```
Naive Bayes Accuracy 72.5454545455
('Correctly Classified Important Emails: ', 72)
('Incorrectly Classified Important Emails: ', 16)
('Could not determine: ', 12)
________________________________________
('Correctly Classified Spam Emails: ', 58)
('Incorrectly Classified Spam Emails: ', 29)
('Could not determine: ', 13)
________________________________________
('Accuracy ', 74.28571428571429)
('Positive Precision ', 0.7128712871287128)
('Positive Recall ', 0.8181818181818182)
('Positive F measure ', 0.761904761904762)
('Negative Precision ', 0.7837837837837838)
('Negative Recall ', 0.6666666666666666)
('Negative F measure ', 0.7204968944099379)

Process finished with exit code 0
```

Fig 5.5: Results using Lemmatizing on Naive Bayes



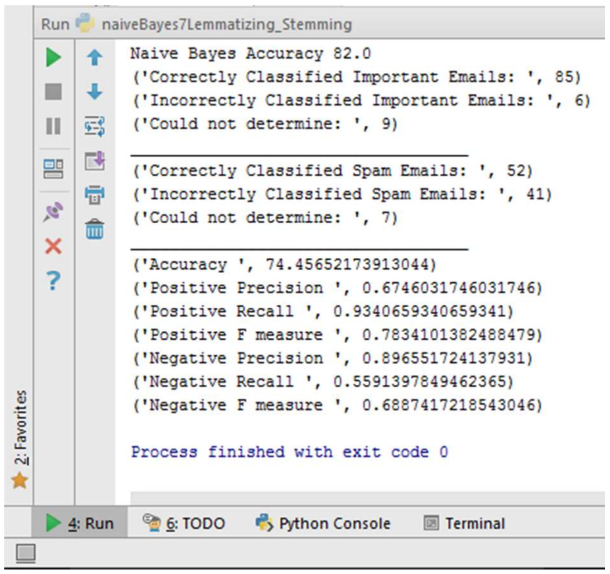
```
Naive Bayes Accuracy 83.2727272727
('Correctly Classified Important Emails: ', 74)
('Incorrectly Classified Important Emails: ', 14)
('Could not determine: ', 12)
________________________________________
('Correctly Classified Spam Emails: ', 58)
('Incorrectly Classified Spam Emails: ', 30)
('Could not determine: ', 12)
________________________________________
('Accuracy ', 75.0)
('Positive Precision ', 0.7115384615384616)
('Positive Recall ', 0.8409090909090909)
('Positive F measure ', 0.7708333333333333)
('Negative Precision ', 0.8055555555555556)
('Negative Recall ', 0.6590909090909091)
('Negative F measure ', 0.7250000000000001)

Process finished with exit code 0
```

Fig 5.6: Results using Stop Words & Stemming on Naive Bayes



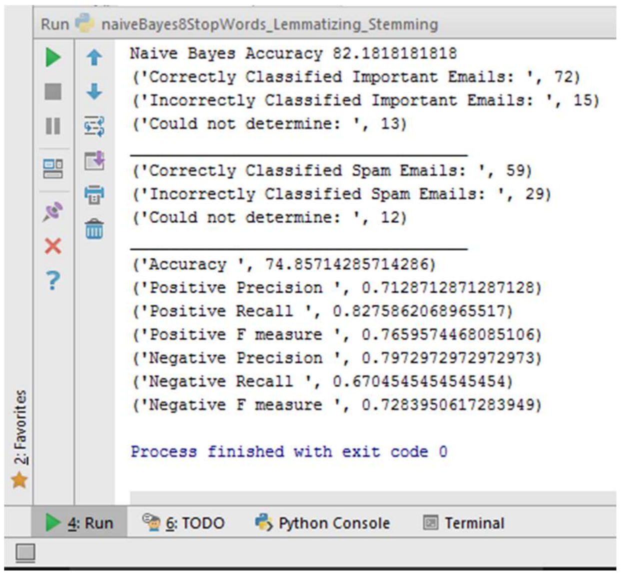
```
Naive Bayes Accuracy 66.5454545455
('Correctly Classified Important Emails: ', 70)
('Incorrectly Classified Important Emails: ', 16)
('Could not determine: ', 14)
________________________________________
('Correctly Classified Spam Emails: ', 67)
('Incorrectly Classified Spam Emails: ', 20)
('Could not determine: ', 13)
________________________________________
('Accuracy ', 79.1907514450867)
('Positive Precision ', 0.7777777777777778)
('Positive Recall ', 0.813953488372093)
('Positive F measure ', 0.7954545454545455)
('Negative Precision ', 0.8072289156626506)
('Negative Recall ', 0.7701149425287356)
('Negative F measure ', 0.788235294117647)

Process finished with exit code 0
```

Fig 5.7: Results using Stop Words & Lemmat- -izing on Naive Bayes



```
Naive Bayes Accuracy 82.0
('Correctly Classified Important Emails: ', 85)
('Incorrectly Classified Important Emails: ', 6)
('Could not determine: ', 9)
_______________________________________
('Correctly Classified Spam Emails: ', 52)
('Incorrectly Classified Spam Emails: ', 41)
('Could not determine: ', 7)
_______________________________________
('Accuracy ', 74.45652173913044)
('Positive Precision ', 0.6746031746031746)
('Positive Recall ', 0.9340659340659341)
('Positive F measure ', 0.7834101382488479)
('Negative Precision ', 0.896551724137931)
('Negative Recall ', 0.5591397849462365)
('Negative F measure ', 0.6887417218543046)

Process finished with exit code 0
```

Fig 5.8: Results using Lemmatizing & Stemm-
-ing on Naive Bayes



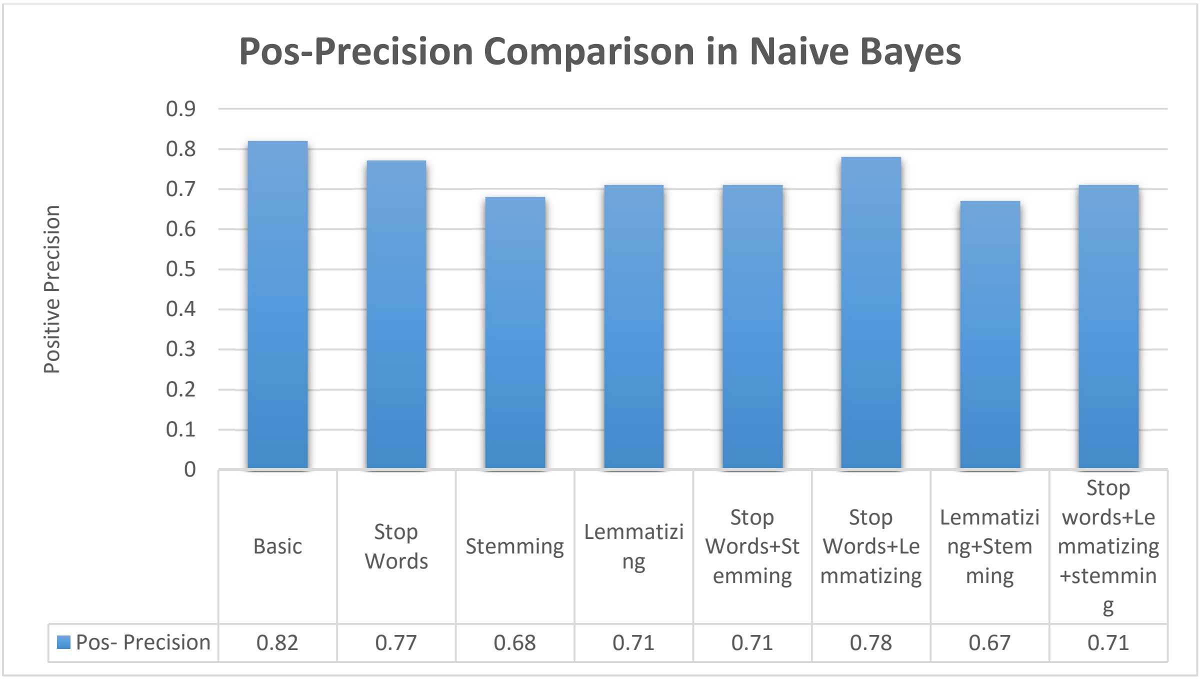
```
Naive Bayes Accuracy 82.1818181818
('Correctly Classified Important Emails: ', 72)
('Incorrectly Classified Important Emails: ', 15)
('Could not determine: ', 13)
_______________________________________
('Correctly Classified Spam Emails: ', 59)
('Incorrectly Classified Spam Emails: ', 29)
('Could not determine: ', 12)
_______________________________________
('Accuracy ', 74.85714285714286)
('Positive Precision ', 0.7128712871287128)
('Positive Recall ', 0.8275862068965517)
('Positive F measure ', 0.7659574468085106)
('Negative Precision ', 0.7972972972972973)
('Negative Recall ', 0.6704545454545454)
('Negative F measure ', 0.7283950617283949)

Process finished with exit code 0
```

Fig 5.9: Results using Stop Words, Lemmatizi-
-ng & Stemming on Naive Bayes

Table 5.2 describes the detail result of precision, recall and F-measure of the test set using 8 combinations of different approach to Naive Bayes Classification. The results are measured for both positive and negative perspective. Figure 5.10, 5.11, 5.12, 5.13, 5.14 and 5.15 shows the comparison among different approaches to calculate precision, recall and F-measure for both positive and negative. Figure 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9 were taken from output window of PyCharm which we used to run our code written in python. They show all of the results in detail. In table 5.2, we can see that among all 8 approaches, basic naive bayes gives highest positive precision 0.82 and Lemmatizing+Stemming gives lowest positive precision 0.67. Then Lemmatizing+Stemming gives the highest positive recall 0.93 and basic naive bayes gives the lowest positive recall 0.75. Stop Words gives the highest positive F-measure 0.81 and Lemmatizing gives the lowest positive F-measure 0.76. We also see that, Lemmatizing+ Stemming gives the highest negative precision 0.90 and basic naive bayes gives the lowest negative precision 0.75. Then basic naive bayes gives highest negative recall 0.82 and Lemmatizing+Stemming gives lowest negative recall 0.56. Lastly, StopWords+Lemmatizing gives the highest negative F-measure 0.79 and both Stemming and Lemmatizing+Stemming gives the lowest negative F-measure 0.69.

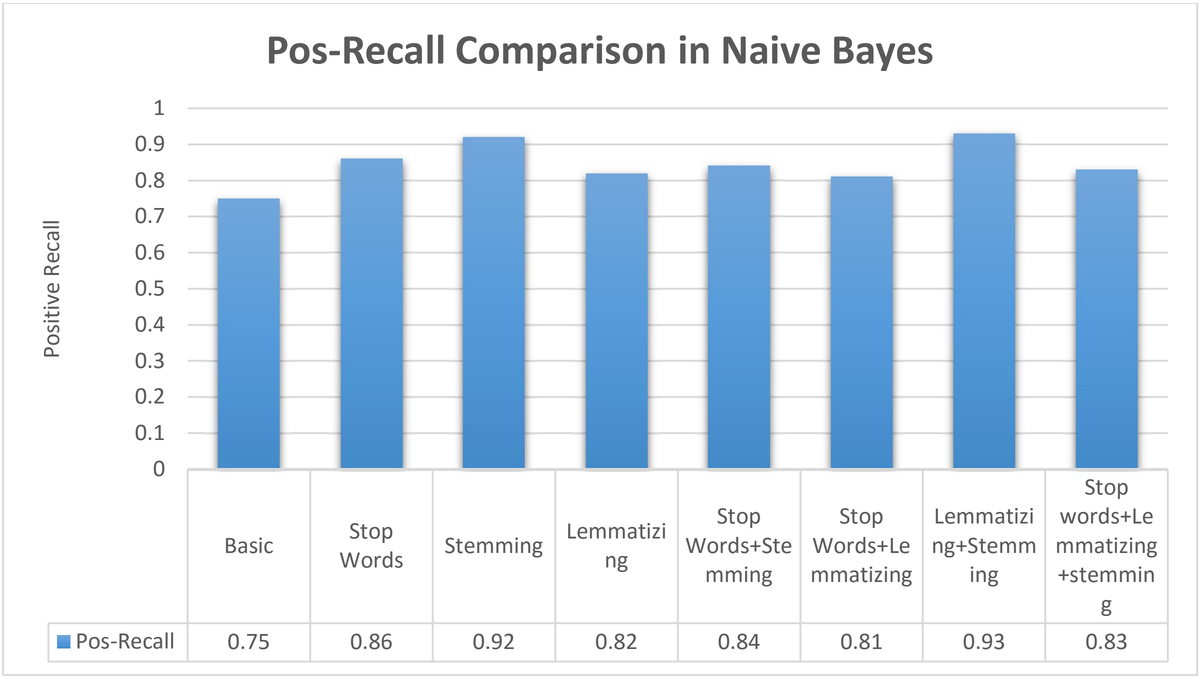
Process	**Accuracy**	**Precision**		**Recall**		**F-Measure**	
		Positive	**Negative**	**Positive**	**Negative**	**Positive**	**Negative**
Basic	78.28	0.82	0.75	0.75	0.82	0.78	0.78
Stop Words	78.65	0.77	0.81	0.86	0.7	0.81	0.75
Stemming	74.46	0.68	0.88	0.92	0.57	0.78	0.69
Lemmatizing	74.29	0.71	0.78	0.82	0.67	0.76	0.72
Stop Words + Stemming	75.00	0.71	0.81	0.84	0.66	0.77	0.73
Stop Words + Lemmatizing	79.19	0.78	0.81	0.81	0.77	0.8	0.79
Lemmatizing + Stemming	74.46	0.67	0.9	0.93	0.56	0.78	0.69
Stop Words + Lemmatizing + Stemming	74.86	0.71	0.8	0.83	0.67	0.77	0.73

Table 5.2: Precision, Recall and F-Measure of Test Set (Naive Bayes in Different Processes)



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
■ Pos- Precision	0.82	0.77	0.68	0.71	0.71	0.78	0.67	0.71

Fig 5.10: Pos-Precision Comparisons (Naive Bayes)



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
■ Pos-Recall	0.75	0.86	0.92	0.82	0.84	0.81	0.93	0.83

Fig 5.11: Pos-Recall Comparisons (Naive Bayes)

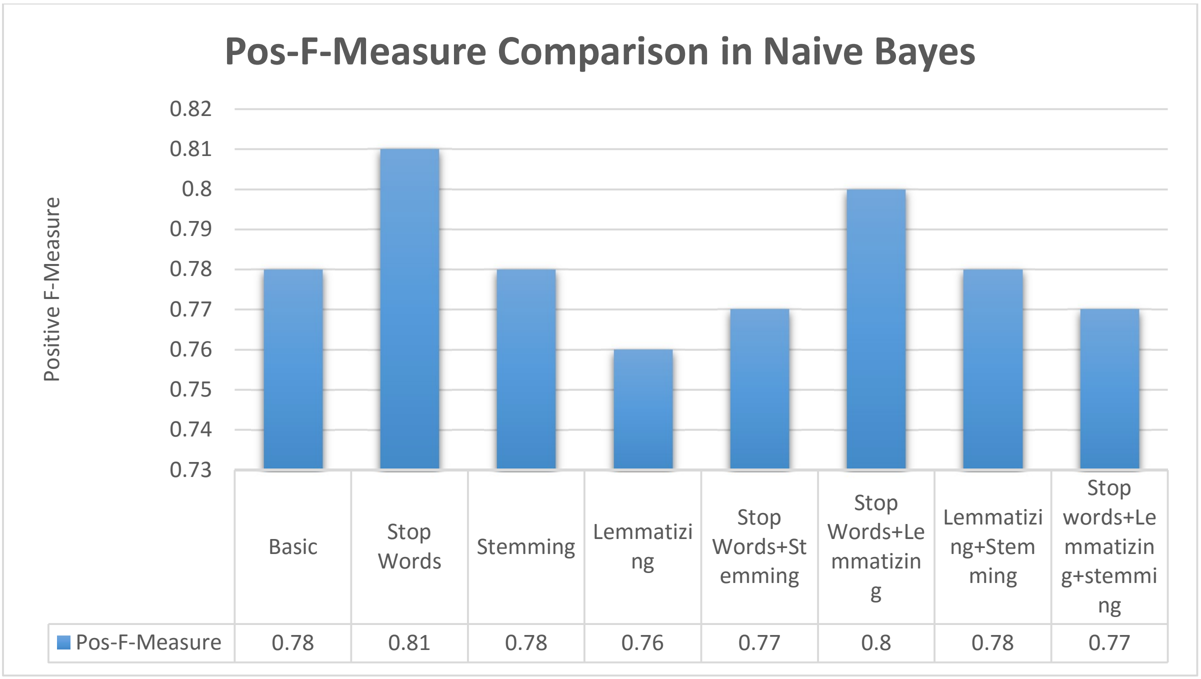


Fig 5.12: Pos-F-Measure Comparisons (Naive Bayes)

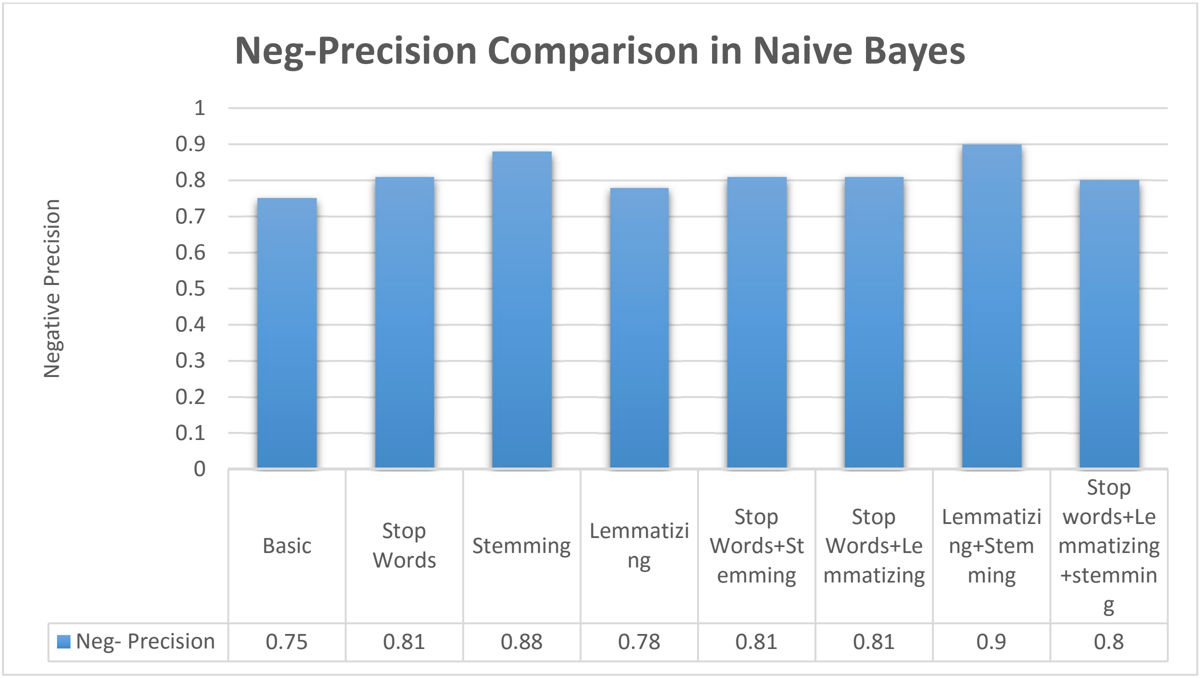
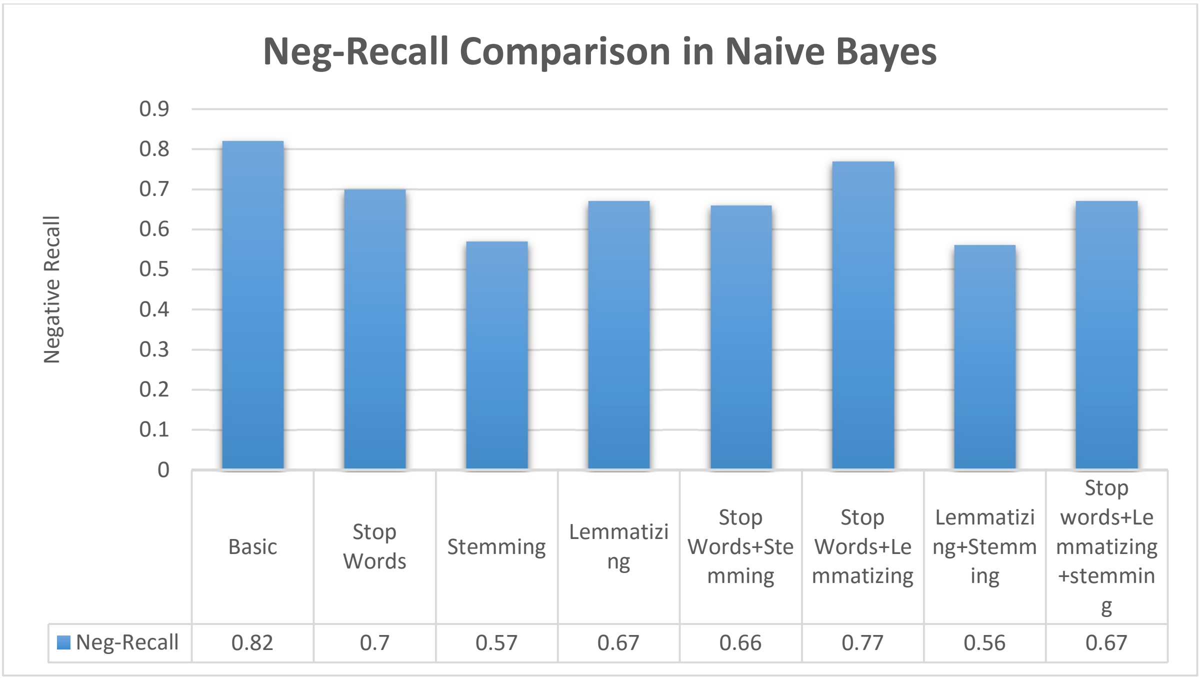
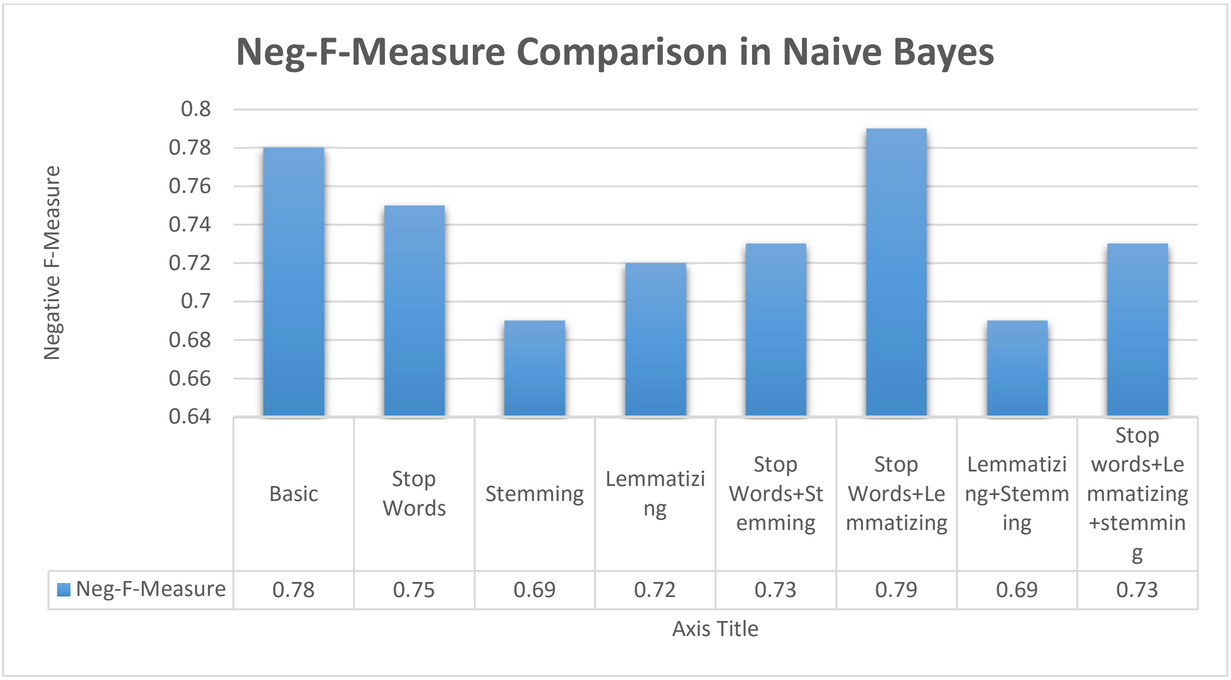


Fig 5.13: Neg-Precision Comparisons (Naive Bayes)



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
🟦 Neg-Recall	0.82	0.7	0.57	0.67	0.66	0.77	0.56	0.67

Fig 5.14: Neg-Recall Comparisons (Naive Bayes)



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
🟦 Neg-F-Measure	0.78	0.75	0.69	0.72	0.73	0.79	0.69	0.73

Fig 5.15: Neg-F-Measure Comparisons (Naive Bayes)

5.1.1.2 Experiment with Hidden Markov Model

As stated earlier classifying with HMM algorithm using basic approach along with various combinations of 3 different processes; Stop Words, Stemming and Lemmatization were used in this experiment. Table 5.3 describes evaluation on test set using basic approach along with various combinations of 3 different processes in HMM algorithm. Figure 5.16 shows the comparison among different approaches used to classify emails based on HMM Algorithm. Figure 5.17, 5.18, 5.19, 5.20, 5.21, 5.22, 5.23 and 5.24 were taken from output window of PyCharm which we used to run our code written in python.

In table 5.3, in total 200 emails were experimented as test set where 100 was important email and 100 was spam email. Some instances were classified correctly, some were classified incorrectly and very few emails remained indeterminate as the possibility of being both important and spam were 50% in these emails. The indeterminate instances were ignored in this case as they could be either important or spam. Using basic HMM approach out of 100 important emails 80 instances were classified correctly, 17 instances were classified incorrectly and 3 instances could not be determined. And out of 100 spams 81 instances were classified correctly, 11 instances were classified incorrectly and 8 instances could not be determined. The total accuracy achieved was 85.19%.

Then again using only stop words, out of 100 important emails 87 instances were classified correctly, 11 instances were classified incorrectly and 2 instances could not be determined. And out of 100 spams 71 instances were classified correctly, 19 instances were classified incorrectly and 10 instances could not be determined. The total accuracy achieved was 84.04%.

Using only stemming out of 100 important emails 87 instances were classified correctly, 10 instances were classified incorrectly and 3 instances could not be determined. And out of 100 spams 91 instances were classified correctly, 7 instances were classified incorrectly and 2 instances could not be determined. The total accuracy achieved was 91.28%.

Process	**Trained Data**	**Test data**	**Correctly Classified instances**		**Incorrectly Classified instances**		**Indeter--minate instances**		**Accuracy (Mail Classifi--cation)**
			Impor--tant	Spam	Impor--tant	Spam	Impor--tant	Spam	
Basic	5500	200	80	81	17	11	3	8	85.19
Stop Words	5500	200	87	71	11	19	2	10	84.04
Stemming	5500	200	87	91	10	7	3	2	91.28
Lemmatizing	5500	200	83	77	15	17	2	6	83.33
Stop Words + Stemming	5500	200	91	74	7	22	2	4	85.05
Stop Words + Lemmatizing	5500	200	86	68	13	23	1	9	81.05
Lemmatizing + Stemming	5500	200	86	89	10	7	4	4	91.15
Stop Words + Lemmatizing + Stemming	5500	200	91	72	7	23	2	5	84.46

Table 5.3: Evaluation on Test Set (Hidden Markov Model in Different Processes)

Using Lemmatizing out of 100 important emails 83 instances were classified correctly, 15 instances were classified incorrectly and 2 instances could not be determined. And out of 100 spams 77 instances were classified correctly, 17 instances were classified incorrectly and 6 instances could not be determined. The total accuracy achieved was 83.33%.

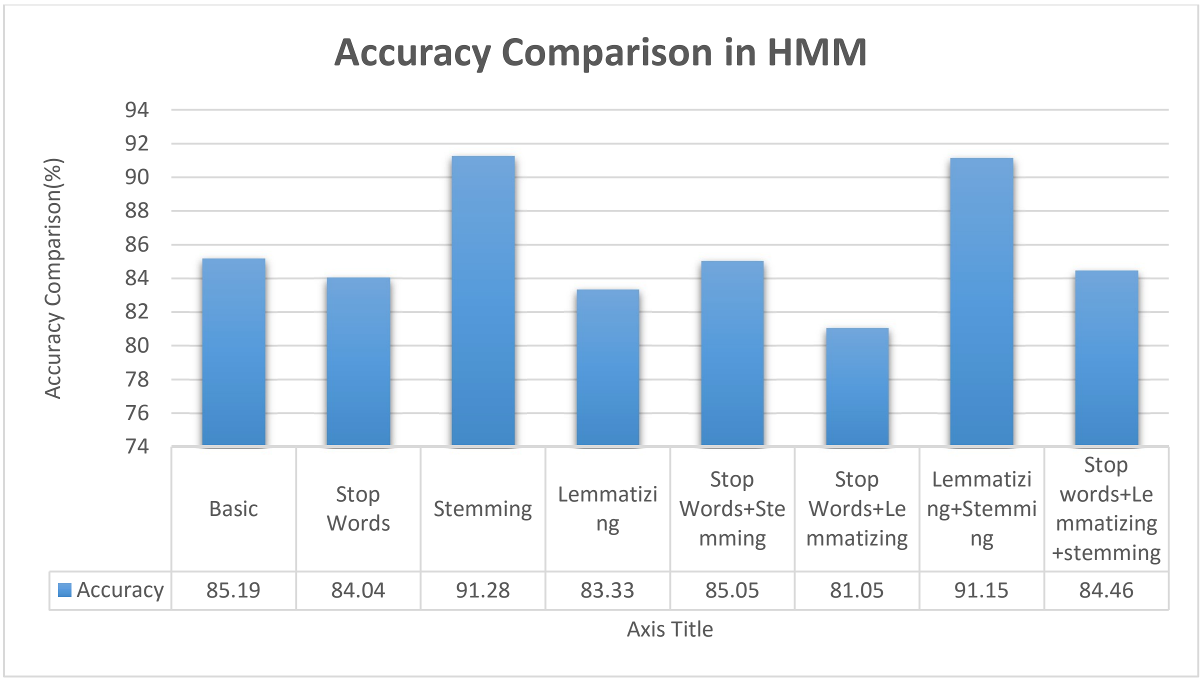
Using both stop words and stemming out of 100 important emails 91 instances were classified correctly, 7 instances were classified incorrectly and 2 instances could not be determined. And out of 100 spams 74 instances were classified correctly, 22 instances were classified incorrectly and 4 instances could not be determined. The total accuracy achieved was 85.05%.

Using both stop words and lemmatizing out of 100 important emails 86 instances were classified correctly, 13 instances were classified incorrectly and 1 instances could not be determined. And out of 100 spams 68 instances were classified correctly, 23 instances were classified incorrectly and 9 instances could not be determined. The total accuracy achieved was 81.05 %.

Using first lemmatizing then stemming out of 100 important emails 86 instances were classified correctly, 10 instances were classified incorrectly and 4 instances could not be determined. And out of 100 spams 89 instances were classified correctly, 7 instances were classified incorrectly and 4 instances could not be determined. The total accuracy achieved was 91.15%.

Lastly using stop words, lemmatizing and stemming out of 100 important emails 91 instances were classified correctly, 7 instances were classified incorrectly and 2 instances could not be determined. And out of 100 spams 72 instances were classified correctly, 23 instances were classified incorrectly and 5 instances could not be determined. The total accuracy achieved was 84.46 %.

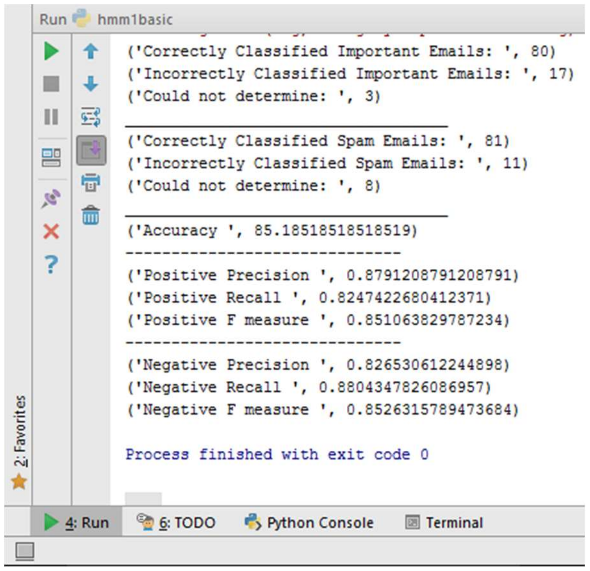
Figure 5.16 shows accuracy comparison among 8 different combinational approach to Hidden Markov Model and it shows that using only stemming gives the best accuracy result which is 91.28%.



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
🟦 Accuracy	85.19	84.04	91.28	83.33	85.05	81.05	91.15	84.46

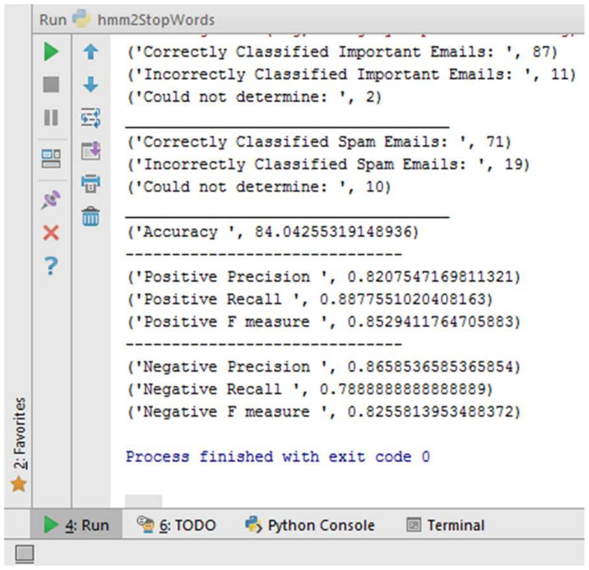
Fig 5.16: Accuracy Comparisons (Hidden Markov Model)

Detailed results of Hidden Markov Model found from different approaches using NLTK in python are shown below-



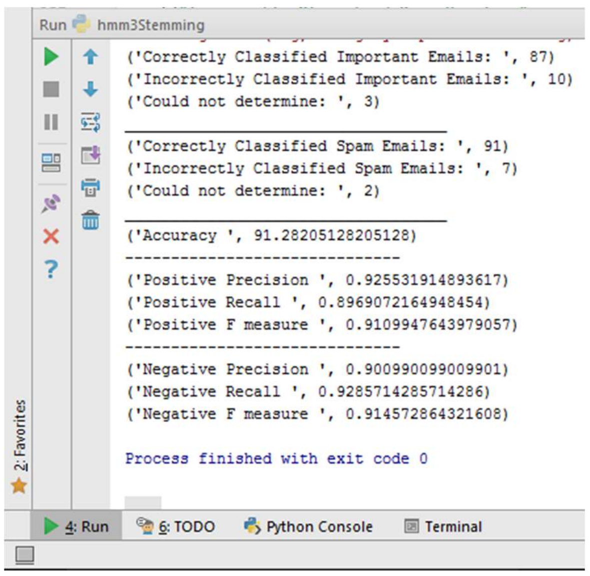
```
('Correctly Classified Important Emails: ', 80)
('Incorrectly Classified Important Emails: ', 17)
('Could not determine: ', 3)
______________________________________
('Correctly Classified Spam Emails: ', 81)
('Incorrectly Classified Spam Emails: ', 11)
('Could not determine: ', 8)
______________________________________
('Accuracy ', 85.18518518518519)
--------------------------------
('Positive Precision ', 0.8791208791208791)
('Positive Recall ', 0.8247422680412371)
('Positive F measure ', 0.851063829787234)
--------------------------------
('Negative Precision ', 0.826530612244898)
('Negative Recall ', 0.8804347826086957)
('Negative F measure ', 0.8526315789473684)

Process finished with exit code 0
```

Fig 5.17: Results using Basic approach on HMM



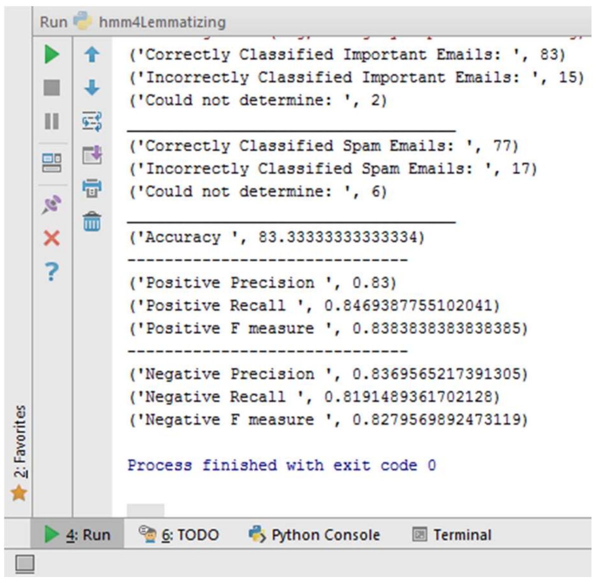
```
('Correctly Classified Important Emails: ', 87)
('Incorrectly Classified Important Emails: ', 11)
('Could not determine: ', 2)
______________________________________
('Correctly Classified Spam Emails: ', 71)
('Incorrectly Classified Spam Emails: ', 19)
('Could not determine: ', 10)
______________________________________
('Accuracy ', 84.04255319148936)
--------------------------------
('Positive Precision ', 0.8207547169811321)
('Positive Recall ', 0.8877551020408163)
('Positive F measure ', 0.8529411764705883)
--------------------------------
('Negative Precision ', 0.8658536585365854)
('Negative Recall ', 0.7888888888888889)
('Negative F measure ', 0.8255813953488372)

Process finished with exit code 0
```

Fig 5.18: Results using Stop Words on HMM



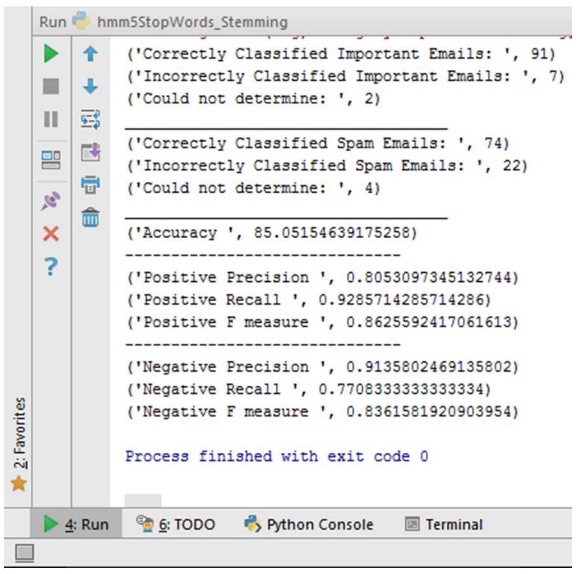
```
('Correctly Classified Important Emails: ', 87)
('Incorrectly Classified Important Emails: ', 10)
('Could not determine: ', 3)
____________________________________
('Correctly Classified Spam Emails: ', 91)
('Incorrectly Classified Spam Emails: ', 7)
('Could not determine: ', 2)
____________________________________
('Accuracy ', 91.28205128205128)
------------------------------
('Positive Precision ', 0.925531914893617)
('Positive Recall ', 0.8969072164948454)
('Positive F measure ', 0.9109947643979057)
------------------------------
('Negative Precision ', 0.900990099009901)
('Negative Recall ', 0.9285714285714286)
('Negative F measure ', 0.914572864321608)

Process finished with exit code 0
```

Fig 5.19: Results using Stemming on HMM



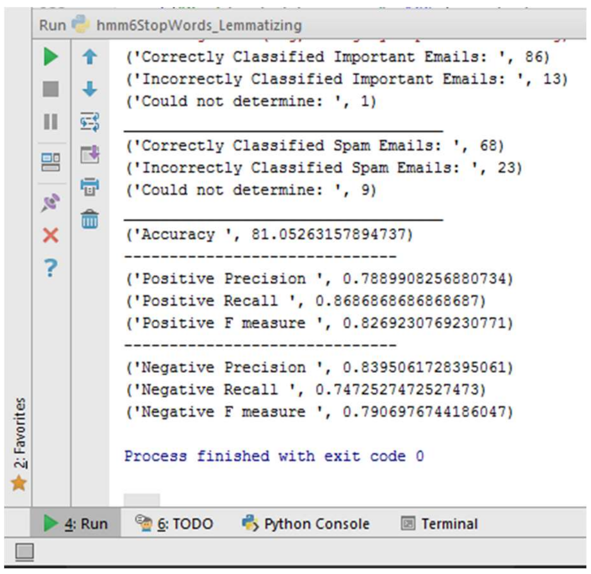
```
('Correctly Classified Important Emails: ', 83)
('Incorrectly Classified Important Emails: ', 15)
('Could not determine: ', 2)
____________________________________
('Correctly Classified Spam Emails: ', 77)
('Incorrectly Classified Spam Emails: ', 17)
('Could not determine: ', 6)
____________________________________
('Accuracy ', 83.33333333333334)
------------------------------
('Positive Precision ', 0.83)
('Positive Recall ', 0.8469387755102041)
('Positive F measure ', 0.8383838383838385)
------------------------------
('Negative Precision ', 0.8369565217391305)
('Negative Recall ', 0.8191489361702128)
('Negative F measure ', 0.8279569892473119)

Process finished with exit code 0
```

Fig 5.20: Results using Lemmatizing on HMM



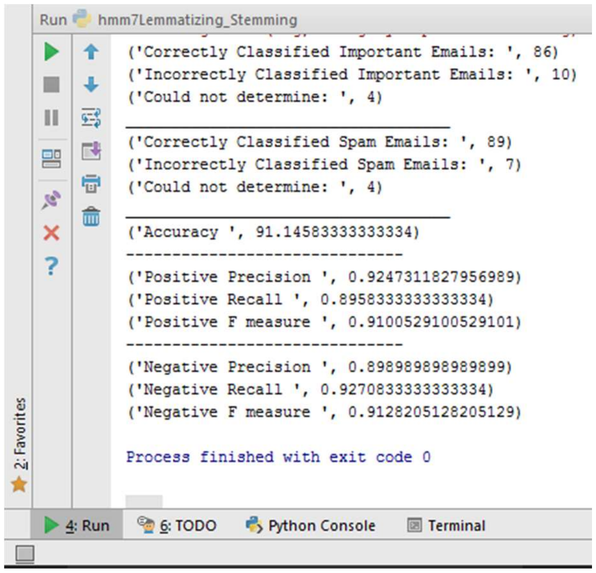
```
('Correctly Classified Important Emails: ', 91)
('Incorrectly Classified Important Emails: ', 7)
('Could not determine: ', 2)
____________________________________
('Correctly Classified Spam Emails: ', 74)
('Incorrectly Classified Spam Emails: ', 22)
('Could not determine: ', 4)
____________________________________
('Accuracy ', 85.05154639175258)
------------------------------
('Positive Precision ', 0.8053097345132744)
('Positive Recall ', 0.9285714285714286)
('Positive F measure ', 0.8625592417061613)
------------------------------
('Negative Precision ', 0.9135802469135802)
('Negative Recall ', 0.7708333333333334)
('Negative F measure ', 0.8361581920903954)

Process finished with exit code 0
```

Fig 5.21: Results using Stop Words & Stemm--ing on HMM



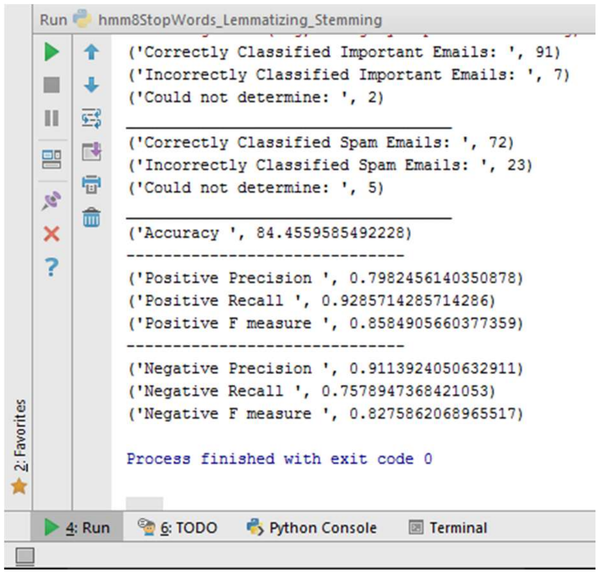
```
('Correctly Classified Important Emails: ', 86)
('Incorrectly Classified Important Emails: ', 13)
('Could not determine: ', 1)
____________________________________
('Correctly Classified Spam Emails: ', 68)
('Incorrectly Classified Spam Emails: ', 23)
('Could not determine: ', 9)
____________________________________
('Accuracy ', 81.05263157894737)
------------------------------
('Positive Precision ', 0.7889908256880734)
('Positive Recall ', 0.8686868686868687)
('Positive F measure ', 0.8269230769230771)
------------------------------
('Negative Precision ', 0.8395061728395061)
('Negative Recall ', 0.7472527472527473)
('Negative F measure ', 0.7906976744186047)

Process finished with exit code 0
```

Fig 5.22: Results using Stop Words & Lemmati--zing on HMM



```
('Correctly Classified Important Emails: ', 86)
('Incorrectly Classified Important Emails: ', 10)
('Could not determine: ', 4)
____________________________________
('Correctly Classified Spam Emails: ', 89)
('Incorrectly Classified Spam Emails: ', 7)
('Could not determine: ', 4)
____________________________________
('Accuracy ', 91.14583333333334)
------------------------------
('Positive Precision ', 0.9247311827956989)
('Positive Recall ', 0.8958333333333334)
('Positive F measure ', 0.9100529100529101)
------------------------------
('Negative Precision ', 0.898989898989899)
('Negative Recall ', 0.9270833333333334)
('Negative F measure ', 0.9128205128205129)

Process finished with exit code 0
```

Fig 5.23: Results using Lemmatizing & Stemm- -ing on HMM



```
('Correctly Classified Important Emails: ', 91)
('Incorrectly Classified Important Emails: ', 7)
('Could not determine: ', 2)
____________________________________
('Correctly Classified Spam Emails: ', 72)
('Incorrectly Classified Spam Emails: ', 23)
('Could not determine: ', 5)
____________________________________
('Accuracy ', 84.4559585492228)
------------------------------
('Positive Precision ', 0.7982456140350878)
('Positive Recall ', 0.9285714285714286)
('Positive F measure ', 0.8584905660377359)
------------------------------
('Negative Precision ', 0.9113924050632911)
('Negative Recall ', 0.7578947368421053)
('Negative F measure ', 0.8275862068965517)

Process finished with exit code 0
```

Fig 5.24: Results using Stop Words, Lemmat- -izing & Stemming on HMM

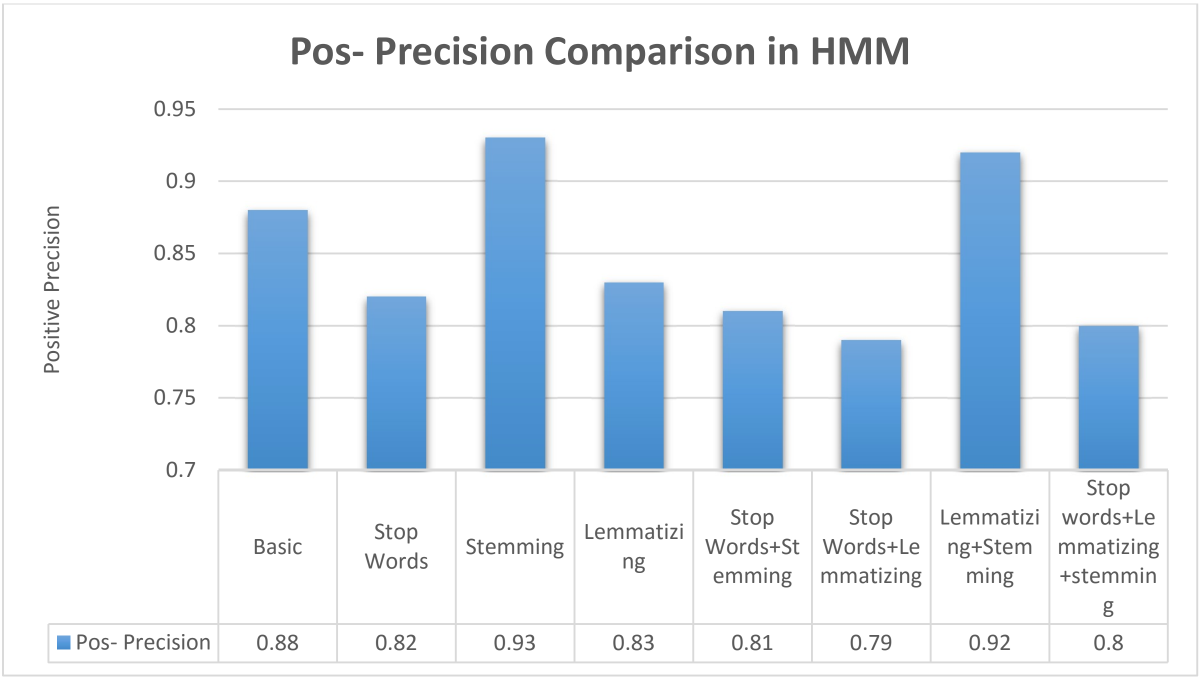
Table 5.4 describes the detail result of precision, recall and F-measure of the test set using 8 combinations of different approach to HMM based Classification. The results are measured for both positive and negative perspective. Figure 5.33, 5.34, 5.35, 5.36, 5.37 and 5.38 shows the comparison among different approaches to calculate precision, recall and F-measure for both positive and negative. Figure 5.17, 5.18, 5.19, 5.20, 5.21, 5.22, 5.23 and 5.24 were taken from output window of PyCharm which we used to run our code written in python. They show all of the results in detail.

In table 5.4, we can see that among all 8 approaches, stemming gives highest positive precision 0.93 and StopWords+Lemmatizing gives lowest positive precision 0.79. Then Stop Words+Stemming and StopWords+Lemmatizing+Stemming gives the highest positive recall 0.93 and basic naive bayes gives the lowest positive recall 0.82. Stemming and Lemmatizing+ Stemming gives the highest positive F-measure 0.91 and StopWords+ Lemmatizing gives the lowest positive F-measure 0.83. We also see that, StopWords+Stemming and StopWords+Lemmatizing+Stemming give the highest negative precision 0.91 and basic HMM gives the lowest negative precision 0.83. Then Stemming and Lemmatizing+Stemming give highest negative recall 0.93 and StopWords+Lemmatizing gives lowest negative recall 0.75.

Lastly, Stemming and Lemmatizing+Stemming give the highest negative F-measure 0.91 and StopWords+ Lemmatizing gives the lowest negative F-measure 0.79.

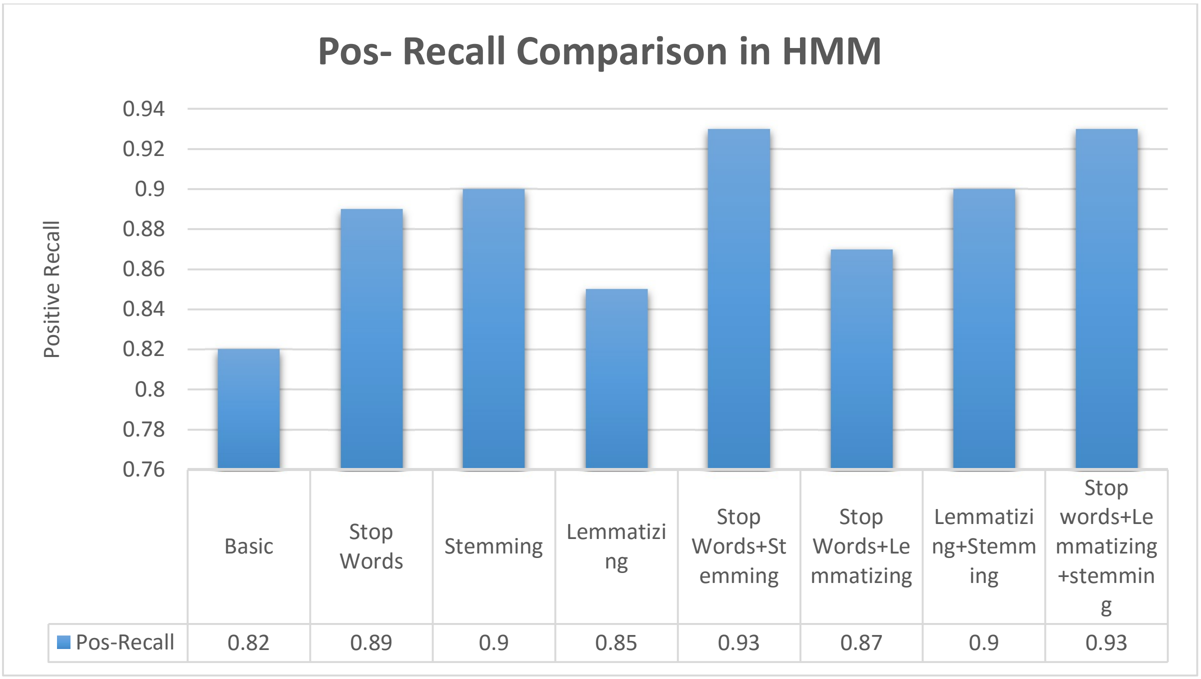
Process	**Accuracy**	**Precision**		**Recall**		**F-Measure**	
		Positive	**Negative**	**Positive**	**Negative**	**Positive**	**Negative**
Basic	85.19	0.88	0.83	0.82	0.88	0.85	0.85
Stop Words	84.04	0.82	0.87	0.89	0.79	0.85	0.83
Stemming	91.28	0.93	0.90	0.90	0.93	0.91	0.91
Lemmatizing	83.33	0.83	0.84	0.85	0.82	0.84	0.83
Stop Words + Stemming	85.05	0.81	0.91	0.93	0.77	0.86	0.84
Stop Words + Lemmatizing	81.05	0.79	0.84	0.87	0.75	0.83	0.79
Lemmatizing + Stemming	91.15	0.92	0.90	0.9	0.93	0.91	0.91
Stop Words + Lemmatizing + Stemming	84.46	0.80	0.91	0.93	0.76	0.86	0.83

Table 5.4: Precision, Recall and F-Measure of Test Set (HMM in Different Processes)



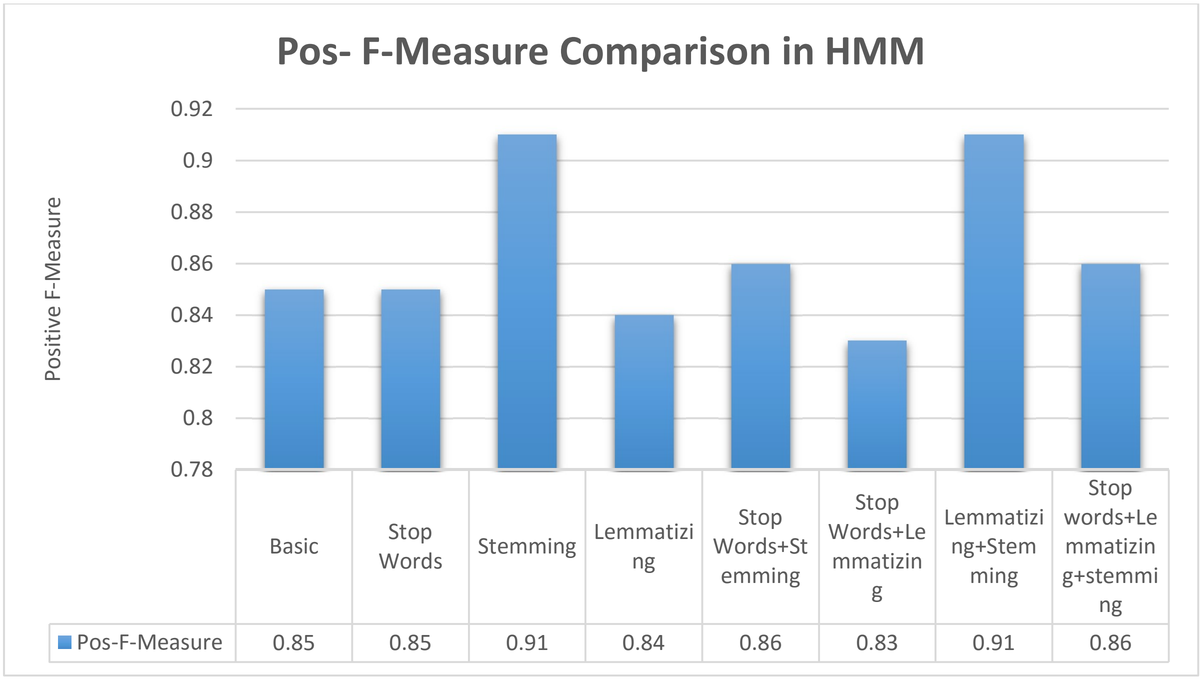
	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
🟦 Pos- Precision	0.88	0.82	0.93	0.83	0.81	0.79	0.92	0.8

Fig 5.25: Pos-Precision Comparisons (HMM)



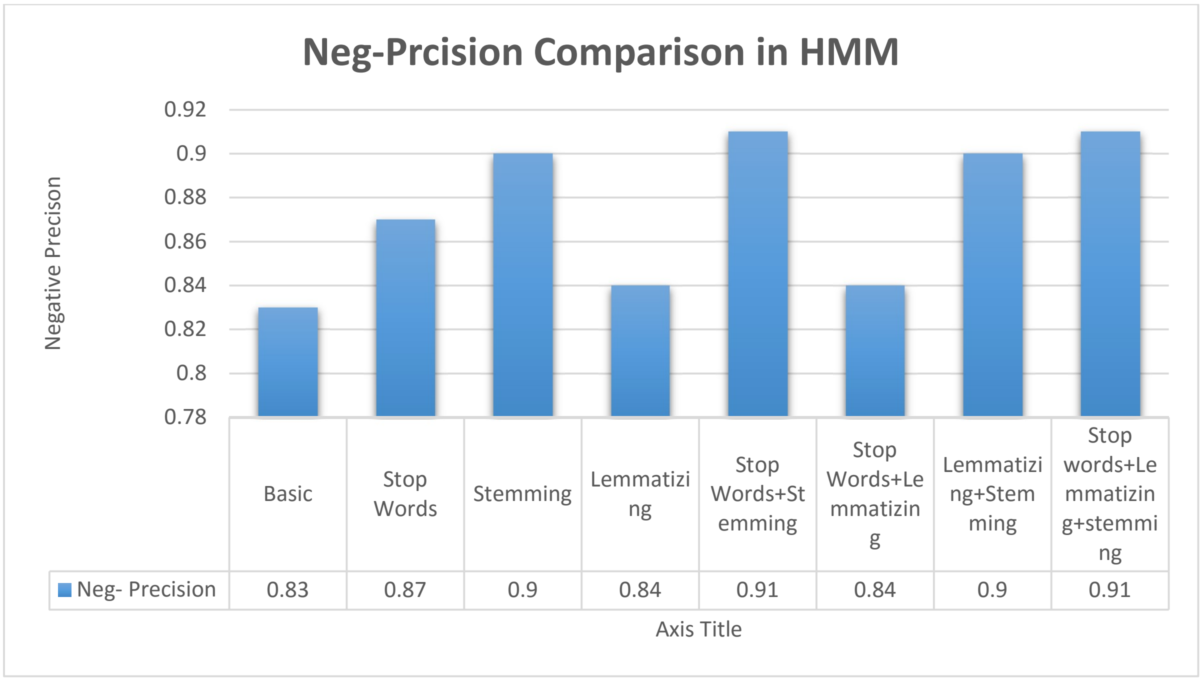
	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
🟦 Pos-Recall	0.82	0.89	0.9	0.85	0.93	0.87	0.9	0.93

Fig 5.26: Pos-Recall Comparisons (HMM)



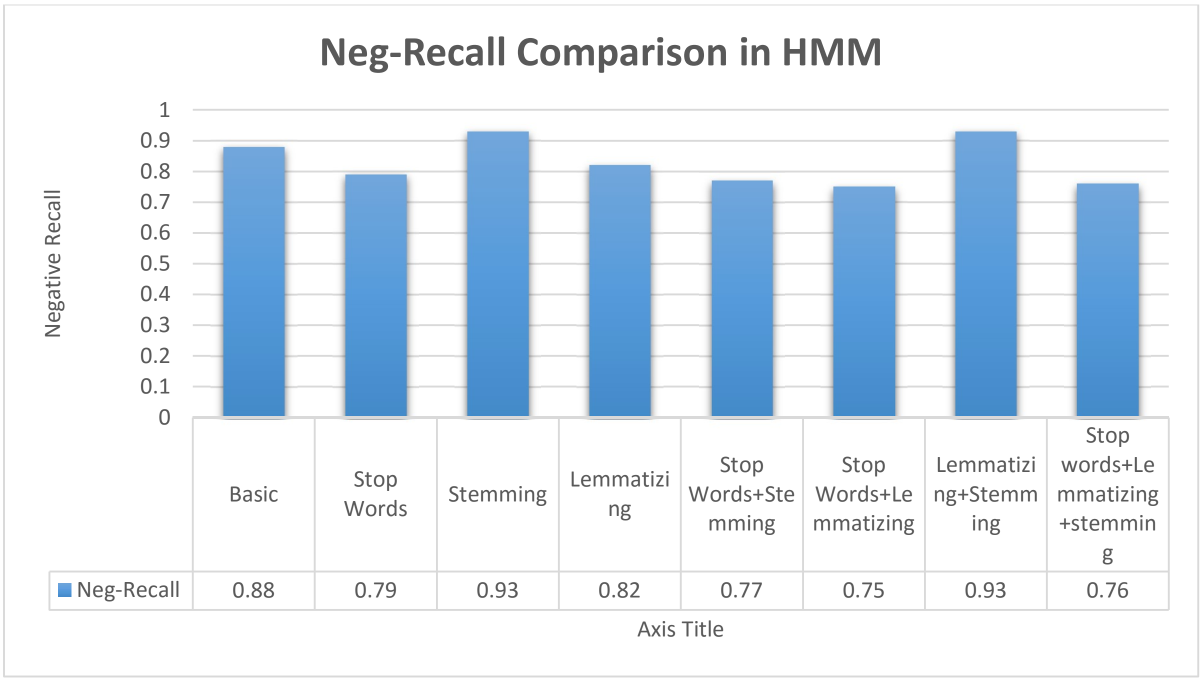
	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
🟦 Pos-F-Measure	0.85	0.85	0.91	0.84	0.86	0.83	0.91	0.86

Fig 5.27: Pos-F-Measure Comparisons (HMM)



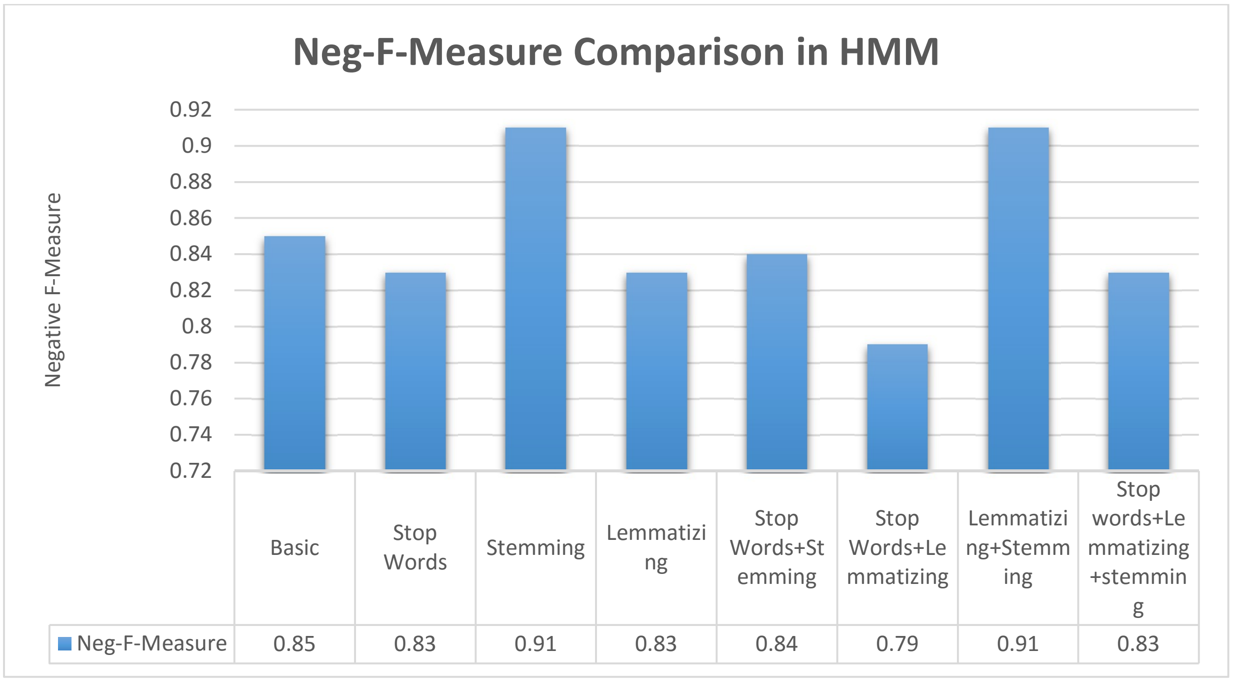
	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
🟦 Neg- Precision	0.83	0.87	0.9	0.84	0.91	0.84	0.9	0.91

Fig 5.28: Neg-Precision Comparisons (HMM)



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
■ Neg-Recall	0.88	0.79	0.93	0.82	0.77	0.75	0.93	0.76

Fig 5.29: Neg-Recall Comparisons (HMM)



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+stemming
■ Neg-F-Measure	0.85	0.83	0.91	0.83	0.84	0.79	0.91	0.83

Fig 5.30: Neg-F-Measure Comparisons (HMM)

5.1.2 Comparison Analysis

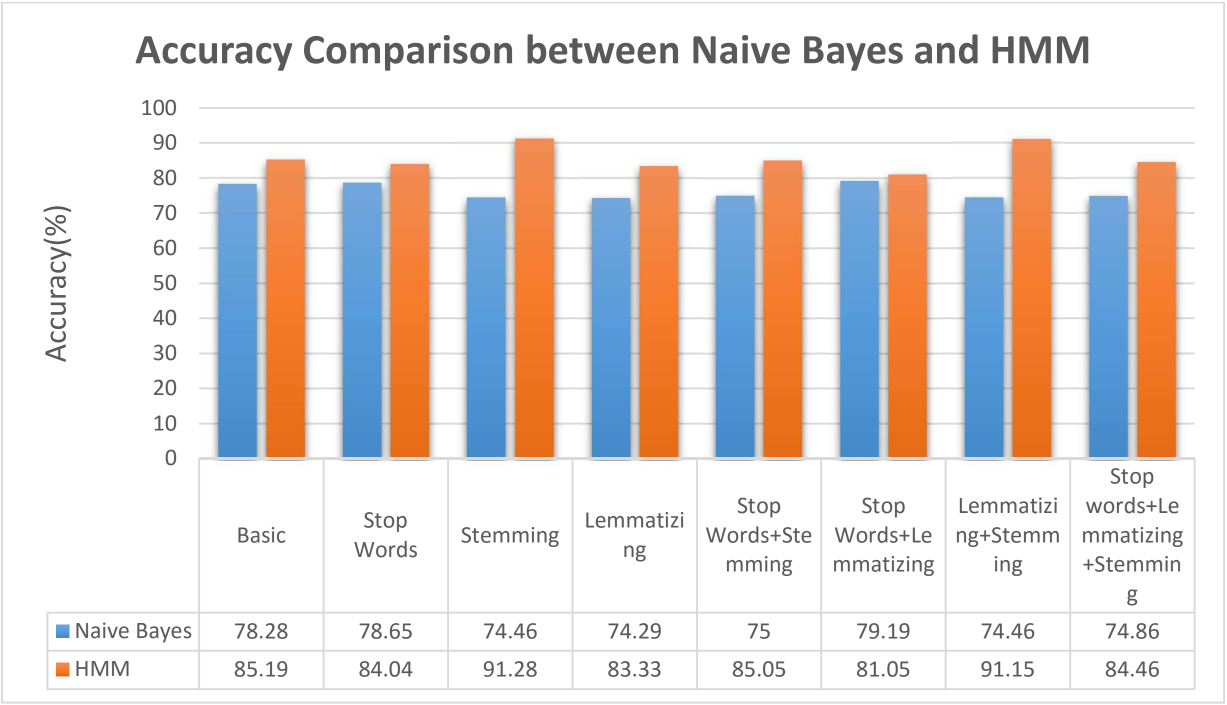
After running both Naive Bayes and HMM algorithm in 8 combinations of 3 different processes along with basic approach we find different classification accuracy for different

Process	**Accuracy (Mail Classification)**	
	Naive Bayes Algorithm	**HMM Algorithm**
Basic	78.28	85.19
Stop Words	78.65	84.04
Stemming	74.46	91.28
Lemmatizing	74.29	83.33
Stop Words + **Stemming**	75.00	85.05
Stop Words + **Lemmatizing**	79.19	81.05
Lemmatizing + **Stemming**	74.46	91.15
Stop Words + **Lemmatizing** + **Stemming**	74.86	84.46

Table 5.5: Accuracy Comparison between Naive Bayes and HMM Algorithm.

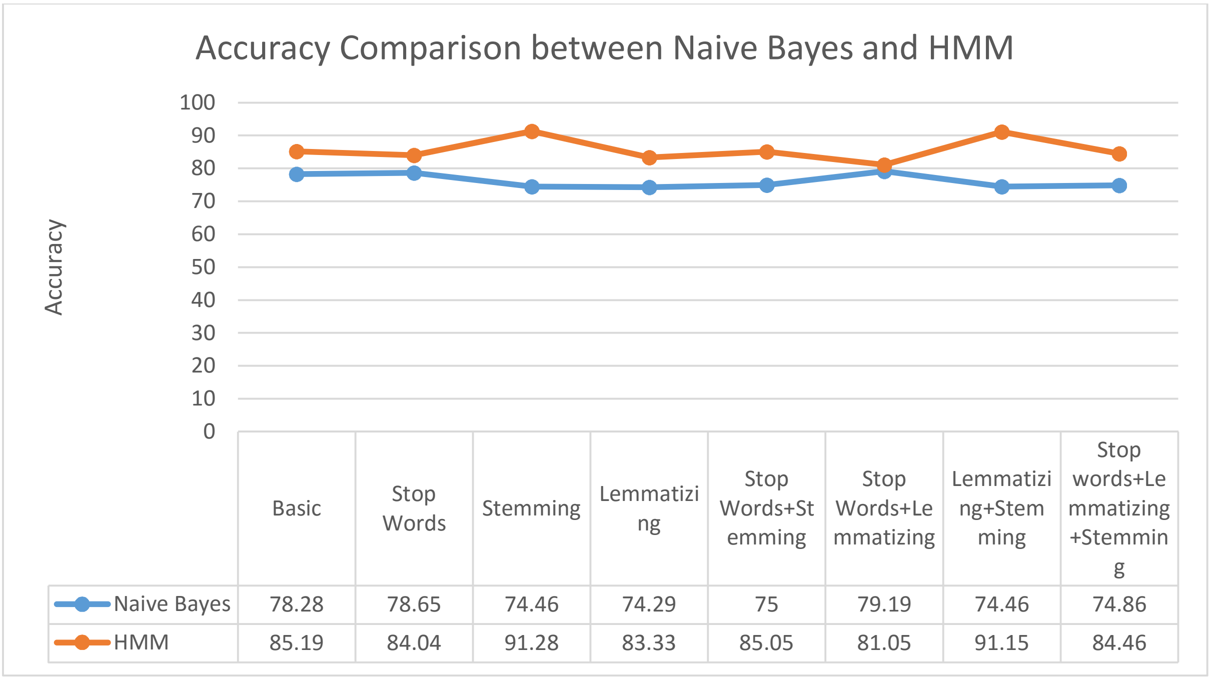
processes. Table 5.5 describes comparison among different accuracies found in different processes of Naive Bayes and HMM algorithm.

Figure 5.31 clearly shows that in every process HMM algorithm gives better accuracy than Naive Bayes algorithm. Though both algorithms get pretty closer when we use Stop Words and Lemmatizing (79.19 & 81.05) but even here HMM algorithm is giving better accuracy than Naive Bayes algorithm. Figure 5.32 represents the graphical representation of Naive Bayes vs HMM algorithm. The dotted points represent the respective accuracies for different approach to both algorithms and the curve of HMM is always higher than the curve of Naive Bayes algorithm. And it is the highest when we use only stemming. So, we can come to the conclusion that to classify emails HMM is always better than Naive Bayes algorithm and using only stemming provides the highest accuracy.



	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing +Stemming
🟦 Naive Bayes	78.28	78.65	74.46	74.29	75	79.19	74.46	74.86
🟧 HMM	85.19	84.04	91.28	83.33	85.05	81.05	91.15	84.46

Fig 5.31: Accuracy Comparison between Naive Bayes and HMM Algorithm



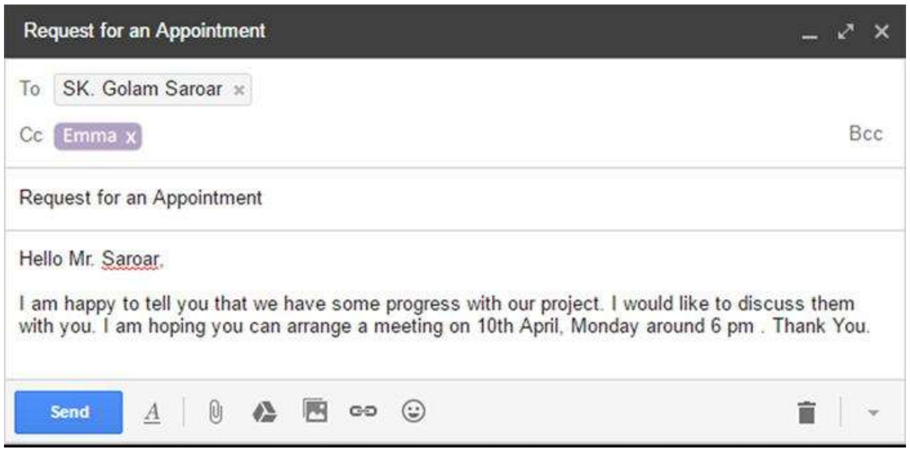
	Basic	Stop Words	Stemming	Lemmatizing	Stop Words+Stemming	Stop Words+Lemmatizing	Lemmatizing+Stemming	Stop words+Lemmatizing+Stemming
Naive Bayes	78.28	78.65	74.46	74.29	75	79.19	74.46	74.86
HMM	85.19	84.04	91.28	83.33	85.05	81.05	91.15	84.46

Fig 5.32: Accuracy Comparison between Naive Bayes and HMM Algorithm

5.2 Scheduler

Even though the underlying algorithm is not that simple, automatic meeting scheduling looks like a piece of cake in plain eyes. If there is no clash, meaning the recipient does not already have a calendar event at that time, our email assistant fixes a meeting at the requested time in the following steps (From here on, we will refer to our email assistant as Emma) :

Step 1a: Sender must Cc Emma, our intelligent email assistant
Step 2a: Emma will know if the email contains a meeting request
Step 3a: Emma will send the sender a confirmation of the appointment.
Step 4a: Appointment will be added on both users' Google calendar.

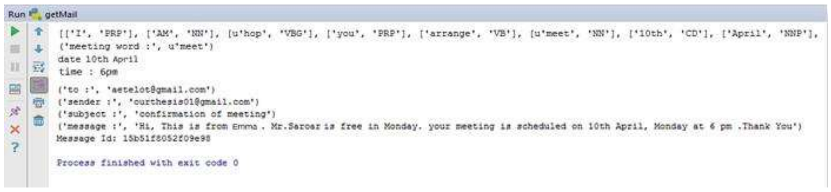


Request for an Appointment

Hello Mr. Saroar,

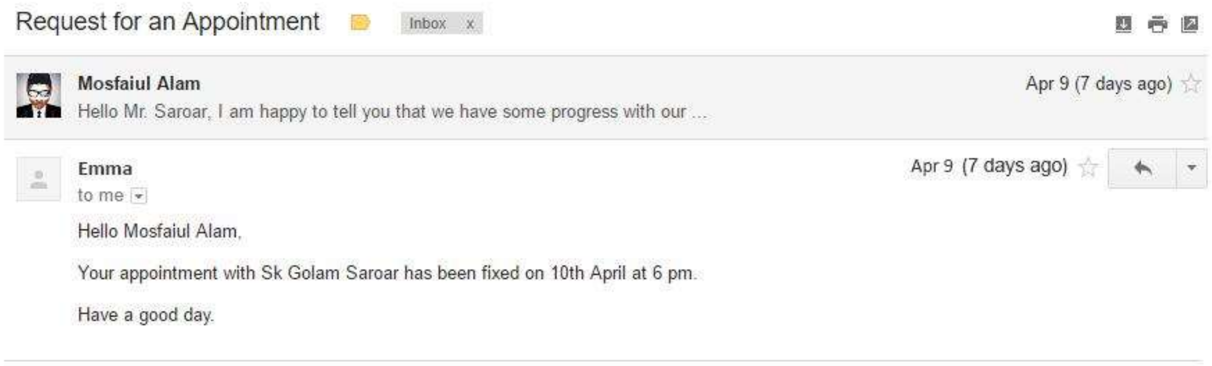
I am happy to tell you that we have some progress with our project. I would like to discuss them with you. I am hoping you can arrange a meeting on 10th April, Monday around 6 pm . Thank You.

Fig 5.33: Step 1a



```
[['I', 'PRP'], ['AM', 'NN'], [u'hop', 'VBG'], ['you', 'PRP'], ['arrange', 'VB'], [u'meet', 'NN'], ['10th', 'CD'], ['April', 'NNP'],
('meeting word :', u'meet')
date 10th April
time : 6pm
('to :', 'aetelot@gmail.com')
('sender :', 'ourthesis01@gmail.com')
('subject :', 'confirmation of meeting')
('message :', 'Hi, This is from Emma . Mr.Saroar is free in Monday. your meeting is scheduled on 10th April, Monday at 6 pm .Thank You')
Message Id: 15b51f8052f09e98

Process finished with exit code 0
```

Fig 5.34: Step 2a



Mosfaiul Alam Apr 9 (7 days ago)

Hello Mr. Saroar, I am happy to tell you that we have some progress with our ...

Emma Apr 9 (7 days ago)

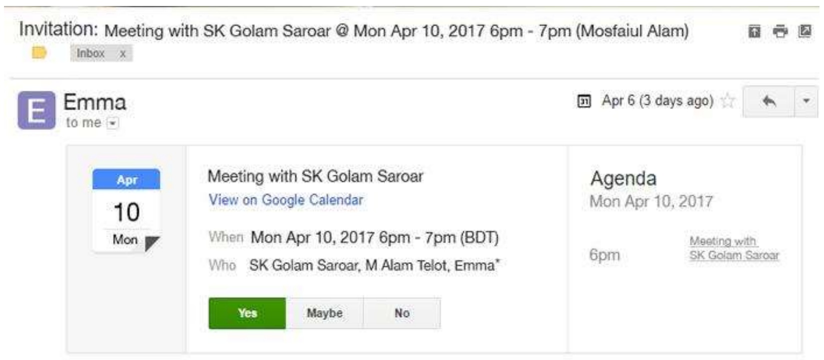
to me

Hello Mosfaiul Alam,

Your appointment with Sk Golam Saroar has been fixed on 10th April at 6 pm.

Have a good day.

Fig 5.34: Step 3a


Invitation: Meeting with SK Golam Saroar @ Mon Apr 10, 2017 6pm - 7pm (Mosfaiul Alam)

Inbox x

Emma
to me

Apr 6 (3 days ago)

Apr 10 Mon

Meeting with SK Golam Saroar
View on Google Calendar

When Mon Apr 10, 2017 6pm - 7pm (BDT)
Who SK Golam Saroar, M Alam Telot, Emma*

Agenda
Mon Apr 10, 2017

6pm Meeting with SK Golam Saroar

Fig 5.34: Step 4a

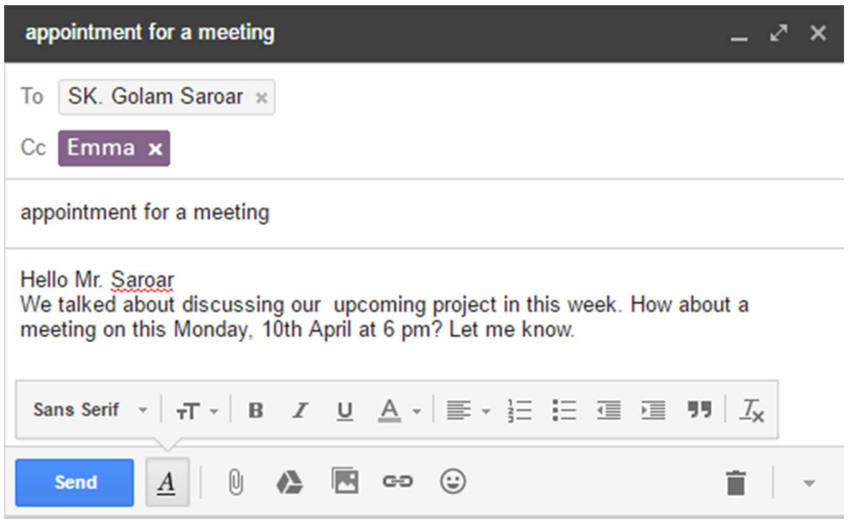
On the other hand, if the recipient already has a calendar event at that time, Emma will notify the sender that the meeting cannot be fixed at the requested time. Sender can reschedule. This happens in the following way:

Step 1b: Another person asks for a meeting at the same time and Cc Emma.

Step 2b: Emma analyzes the email and extracts the date and time of meeting request.

Step 3b: Emma checks the recipient's Google calendar and finds out s/he already has a meeting on that time.

Step 4b: Emma notifies the sender that the meeting cannot be fixed at that time. But s/he can reschedule if s/he wants.


appointment for a meeting

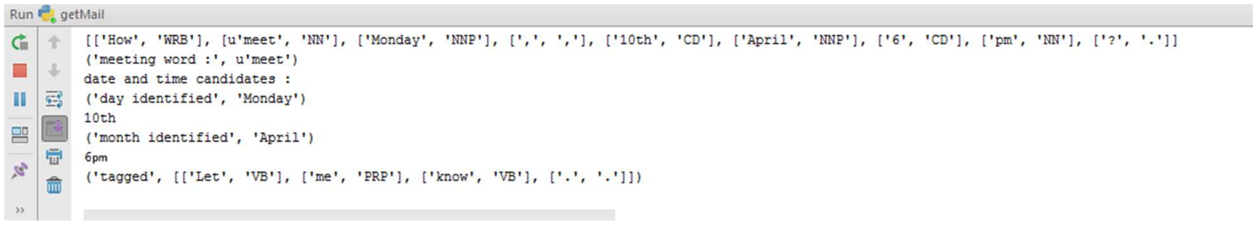
To SK. Golam Saroar

Cc Emma

appointment for a meeting

Hello Mr. Saroar
We talked about discussing our upcoming project in this week. How about a meeting on this Monday, 10th April at 6 pm? Let me know.

Fig 5.35: Step 1b



```
[['How', 'WRB'], [u'meet', 'NN'], ['Monday', 'NNP'], [',', ','], ['10th', 'CD'], ['April', 'NNP'], ['6', 'CD'], ['pm', 'NN'], ['?', '.']]
('meeting word :', u'meet')
date and time candidates :
('day identified', 'Monday')
10th
('month identified', 'April')
6pm
('tagged', [['Let', 'VB'], ['me', 'PRP'], ['know', 'VB'], ['.', '.']])
```

Fig 5.36: Step 2b

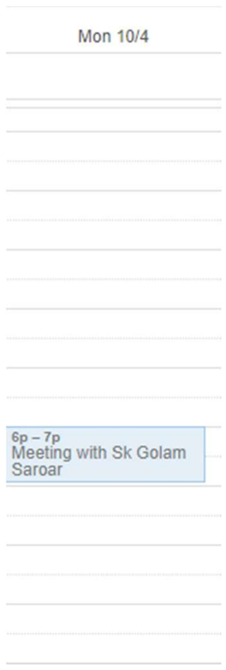
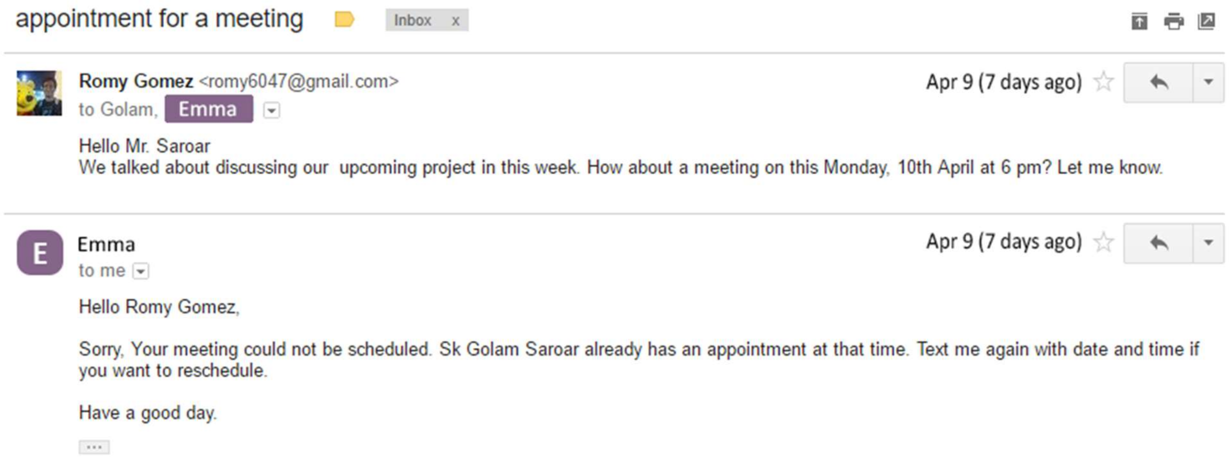


Fig 5.37: Step 3b



appointment for a meeting

Romy Gomez <romy6047@gmail.com>
to Golam, Emma

Hello Mr. Saroar
We talked about discussing our upcoming project in this week. How about a meeting on this Monday, 10th April at 6 pm? Let me know.

Emma
to me

Hello Romy Gomez,

Sorry, Your meeting could not be scheduled. Sk Golam Saroar already has an appointment at that time. Text me again with date and time if you want to reschedule.

Have a good day.

Fig 5.38: Step 4b

CHAPTER 6

Conclusion

We have come a long way from where we started our research a year ago. We like to think we have accomplished a great deal of things, but also acknowledge that we still have many areas to improve. We faced many difficulties over the course of the research, most of which we have successfully overcome. There remains a few with which we did not find a way to work around. Such limitations forced us to adapt to them and re-evaluate our methodologies. We are confident that we have done it well. We have also discovered the scope for further improvement and extension of our work. In this final chapter, we have talked about the difficulties and future plan before we drew a conclusion to this report.

6.1 Difficulties

Email classification constitutes the most part of our work. Our classifier algorithms, both Naive Bayes and HMM, work in a way that can categorize emails in any number of classes like important, urgent, work, social, promotions, forums, spam and a hundred more. Yet we have worked on binary classification - important and spam. It is entirely because of the scarcity of dataset. Our algorithms largely depend on pre-classified dataset. Among all the resources we could get our hands on, we only found email dataset for important and spam emails. We could not train our algorithms on more data also because our working computers did not have the configuration to work with more than 5500 emails. If we could increase our trainset, we believe we would achieve better accuracy. For HMM, we had to use an older version of *sklearn* because the newest version did not support the *hmm* python module we have used. For a very few emails, our algorithms could not categorize them into any class due to the nature of those emails - we found equal number of important and spam words in those emails. Another difficulty that we faced while implementing HMM is precisely calculating emission probability and transmission probability of 5300 words. As we used Google's calendar API to work with user's daily events, our scheduler algorithm is not compatible with any other event manager service.

6.2 Future Plan

For any research, there is always room for improvement. Ours is not an exception of that. While we have marked the end of our research for the time being, we have also pointed out some areas where this research can be stretched:

1. **Other Algorithms:** Other probabilistic graphical model like conditional random field (CRF), maximum entropy etc. can be used to try to achieve better accuracy than HMM.

2. **More Analysis Means Better Accuracy** : System will not only analyze text but also contents of attached files such as pdf, txt, ppt. Computers with better configuration should be used in order to be able to work with more data.

3. **No Limit on Categories:** Currently we are categorizing an email based on two categories- spam and important. In future we want to incorporate more categories such as social, family, educational, research, technology etc. as well as make it possible for users to create their own categories. We will depend on email dataset for the former but the latter can be

achieved rather easily. A user will define the domain for his customized category. For instance, a university professor can categorize the emails he receives from the university in a different section while he categorizes the emails from different conferences in another section.

4. **Smarter Email Assistant:** The system will use past email interactions to recover relevant information for the user to assist in writing their reply. It will also try to understand user's state of mind to generate reply. The system won't be confined in scheduling meetings, it will act as humanly as possible to carry out a general conversation via email.

6.3 Conclusion

In summary, we propose a comparative approach to email classification using Naive Bayes Classifier and HMM. We categorize emails by considering only text part from body of the message. Because we consider relative words and sentences as feature. After running the same variants on both the algorithms, we compared the results and used HMM for classification because it gave better accuracy. Along with email classification, we have also showed how an AI based meeting scheduler can appoint meetings automatically through emails without harming our privacy and serve as our convenience by saving our valuable time. The structure of our research has been built in such a way that with proper dataset and minor altercation it can work to classify texts in any number of categories.

APPENDIX A

Appendix A

A.1 Machine Learning

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed [6]. When exposed to new data, computer programs are enabled to learn, grow, change, and develop by themselves. We are already seeing Machine Learning embedded in many services like Gmail, Search, Maps. In Gmail, Priority Inbox automatically identifies our important incoming messages and separates them out from everything else. It learns over time what is important to us, and what isn't. Smart Reply is another Inbox feature that suggests up to three responses based on the emails one gets. For those emails that only need a quick response, it can take care of the thinking and save time spent typing. The responses get better over time as the system learns.

There are three types of machine learning: [6]

Supervised: Datasets are provided which are used to train the machine and get the desired outputs.

Unsupervised: No datasets are provided, instead the data is clustered into different classes.

Semi-supervised: Some data is labeled but most of it is unlabeled and a mixture of supervised and unsupervised techniques can be used.

A.1.1 Supervised

The majority of practical machine learning uses supervised learning. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. In other words, supervised learning is where there is a set of input (x) and output variables (y) and an algorithm is used to learn the mapping function from the input to the output:

$$y = f(x) \ldots \ldots \ldots (A1)$$

Main objective of supervised learning is to approximate the mapping function so well that when there is new input data (x), we can predict the output variables (y) for that data. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process.

Classification is one type of Supervised Machine Learning. Inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more of these classes. Spam filtering is an example of classification, where the inputs are email messages and the classes are “important” and "spam".

A.1.2 Unsupervised

Sometimes there is no corresponding output variables for input data. It is called unsupervised learning. Unsupervised learning can be seen as a type of machine learning algorithm used to draw deduction from datasets consisting of input data with unlabeled responses.

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning, there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the structure in the data.

The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

A.1.3 Semi-Supervised

When there is a large amount of input data (x) but only some of the data is labeled (y), it is called semi-supervised machine learning.

Many real world machine learning problems fall into this area. This is because it can be expensive as well as time-consuming to label very large amount of data. Often it may require access to domain experts whereas unlabeled data is cheap and easy to collect and store.

Semi-supervised learning techniques are used with a view to discovering and learning the structure in the input variables.

Naive Bayes Classifier is a supervised machine learning algorithm. HMM can be both supervised and unsupervised. In our research, we have implemented HMM using hidden and observed state sequences which is supervised learning. But other techniques of HMM that use Viterbi algorithm and Baum-Welch algorithm fall into unsupervised learning category.

A.2 Bayesian Classification

The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. It assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes. It can solve diagnostic and predictive problems.

Bayesian classification provides practical learning algorithms where prior knowledge and observed data can be combined. Bayesian Classification provides a useful perspective for understanding and evaluating many learning algorithms. It calculates explicit probabilities for hypothesis and it is robust to noise in input data.

Bibliography

[1] Aery, M., & Chakravarthy, S. (2005). eMailSift: eMail classification based on structure and content. *Data Mining, Fifth IEEE Int., 2005.* IEEE. doi:10.1109/ICDM.2005.58

[2] Islam, R., & Zhou, W. (2007). Email Categorization Using Multi-stage Classification Technique. *Parallel and Distributed Computing, Applications and Technologies, 2007. PDCAT '07. Eighth International Conference on,* 2007. IEEE. doi:10.1109/PDCAT.2007.71

[3] Wang, X., & Cloete, I.A.N. (2005). Learning to classify email: a survey. *Proceedings of the international conference on machine learning and, cybernetics, 2005, 9,* 18-21.

[4] Klimt, B., & Yang, Y. (2004). The Enron Corpus: A New Dataset for Email Classification Research. *Boulicaut JF., Esposito F., Giannotti F., Pedreschi D. (eds) Machine Learning: ECML 2004. ECML 2004.* Springer, Berlin, Heidelberg. *Lecture Notes in Computer Science, 3201.*

[5] Bhat, V. H., Malkani, V. R., Shenoy, P. D., Venugopal, K. R., & Patnaik, L. M. (2011). Classification of email using BeaKS: Behavior and keyword stemming. *TENCON 2011 - 2011 IEEE Region 10 Conference*, Bali, 2011. IEEE. 1139-1143. doi: 10.1109/TENCON.2011.6129290

[6] Teli, S. P., & Biradar, S. (2014). Effective Email Classification for Spam and Non-Spam. *International Journal of Advanced Research in Computer Science and Software Engineering, 4*(6).

[7] Yitagesu1, M. E., & Tijare, M. (2016). Email Classification using Classification Method. *International Journal of Engineering Trends and Technology (IJETT), 32*(3), 142.

[8] Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. 3-24

[9] Jaswal, V., & Sood, N. (2013). Spam Detection System Using HMM. *International Journal of Advanced Research in Computer Science and Software Engineering. 3*(6)

[10] Carpinter, J., & Hunt, R. (2006). Tightening the net: a review of current and next generation spam filtering tools. *Computers and Security. 25*(8)*,* 566-578. doi:10.1016/j.cose.2006.06.001,

[11] Kidmose, E. (2014). Botnet detection using HMMs Master Thesis Networks and Distributed System. *Aalborg University Journal*.88

[12] Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

[13] Jivani, A. G. (2011). A Comparative Study of Stemming Algorithms. *International Journal of Computer Technology and Applications. 2*(6), 1930-1938.

[14] BenHassine, A., & TB, H. (2007). An agent-based approach to solve dynamic meeting scheduling problems with preferences. *Eng Appl Artif Intell.* 20, 857–873.

[15] Modi, PJ., Veloso M., Smith SF., & Oh, J. (2004). CMRadar: a personal assistant agent for calendar management. *Proceedings of the 19th national conference on artificial intelligence,* San Jose, California, 1020–1021.

[16] Jackson, P., & Moulinier, I. (2002). *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization.* Amsterdam/Philadelphia: John Benjamins Publishing Company.

[17] Gharamani, Z. (2001). An Introduction to Hidden Markov Models and Bayesian Networks. *Journal of Pattern Recognition and Artificial Intelligence. 15*(1), 9-42.

[18] Saraiya, S. U., & Desai, N. (2015). Content Based Categorization of E-Mail using Hidden Markov Model Approach. *IEEE International Conference on Advances in Engineering and Technology-ICAET 2014,* Tamil-Nadu, Chennai. IEEE. doi: 10.13140/RG.2.1.3070.9602

[19] Pedregosa, F., Varoquaux, G. et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research. 12,* 2825-2830.

[20] Naive-Bayes Classification Algorithm [Online]. Accessed September 2016. Retrieved from http://software.ucv.ro/~cmihaescu/ro/teaching/AIR/docs/Lab4-NaiveBayes.pdf

[21] Python Package Index [Online]. Accessed January 2017.
Retrieved from https://pypi.python.org/pypi/scikit-learn/0.16.1

[22] Enron Email Dataset [Online]. Accessed June 2016.
Retrieved from https://www.cs.cmu.edu/~./enron/

[23] IMAP4 protocol client. Accessed March 2017.
Retrieved from https://docs.python.org/2/library/imaplib.html

[24] Classification: Naïve Bayes Classifier Evaluation. Accessed April 2017.
Retrieved from http://wwwis.win.tue.nl/~tcalders/teaching/datamining09/slides/DM09-02-Classification.pdf