# OPTIMIZING THE NAÏVE BAYES SPAM EMAIL CLASSIFIER

**Research Question:**

**“How can the accuracy of the Naïve Bayes spam email classifier be maximized?”**

3817 words

# TABLE OF CONTENTS:

# INTRODUCTION

## THE PROBLEMS CAUSED BY SPAM EMAILS

Spam emails are defined as unsolicited messages that are sent in bulk via electronic mail. The content of spam emails ranges from phishing scams to pornographic content to the sale of drugs and other illegal goods. Today, spam emails account for about 45% of total global email traffic.[1] The other 55% are ham (genuine) emails.

Spam emails are of various types. One type of spam emails that tend to fill up email user inboxes are advertisements by scammers. These are the most common type of spam emails and aim at selling ingenuine products to customers through false advertising. Another popular type of spam emails propagate Phishing scams in which scammers attempt to extract sensitive information such as usernames, passwords and bank account details from email users. Surprisingly, in 2016, 3 out of 4 companies fell for phishing scams.[2] Spam emails sometimes also contain links to malicious websites that make the computer system vulnerable to malware which is a program that disrupts a computer system's operations, gathers sensitive information from the computer system or gains complete or partial access to the computer system's local files.[3]

Despite anti-spam software and spam filters in place, spam emails continue to have detrimental effects on electronic mail users worldwide. To understand the problem caused in a real-life context, I interviewed the director, Mr. Pratik Shah, of a local business, Shivji Valji

[1] "15 Outrageous Email Spam Statistics That Still Ring True in 2018." RSS. Accessed July 15, 2018. https://www.propellercrm.com/blog/email-spam-statistics.
[2] "15 Outrageous Email Spam Statistics That Still Ring True in 2018." RSS. Accessed July 15, 2018. https://www.propellercrm.com/blog/email-spam-statistics.
[3] "Malware Definition – What Is It and How to Remove It." Malwarebytes. Accessed December 08, 2018. https://www.malwarebytes.com/malware/.

Tarpaulins Pvt Ltd. [4] He said that the problem of spam emails cost his company heavily in a number of ways. For example, by reducing employee productivity, having to invest in anti-malware soft-wares for the computers and wasting email storage space. It also led to slower internet and networking speeds as continuous transmission of spam emails wasted network bandwidth.

Thus, there is a need for more efficient spam filters.

## THE NEED FOR EFFICIENT SPAM EMAIL FILTERS

Most email service providers today use machine learning algorithms to filter out spam emails from a user's inbox. Gmail, for example, uses artificial neural networks along with several other rule-based filters to identify spam emails. Artificial neural networks are a form of machine learning modelled after biological neural networks found in the brain of animals in which computer systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.[5] Due to such an advanced filtering mechanism, Gmail is able to filter out spam emails with an accuracy of 99.9%.[6]

However, not all email service providers can employ such advanced machine learning algorithms because they are difficult to program. Also, such filtration techniques require immense processing power which make them unfeasible to be employed by smaller email service providers or individual companies whose computational abilities are limited.

In the same interview with Mr. Shah, he also stated that his current email service provider did not facilitate accurate spam email filtration and as a result, his employee's inboxes often filled

[4] Appendix 1
[5] "What Is an Artificial Neural Network (ANN)? - Definition from Techopedia." Techopedia.com. Accessed December 08, 2018. https://www.techopedia.com/definition/5967/artificial-neural-network-ann.
[6] Metz, Cade. "Google Says Its AI Catches 99.9 Percent of Gmail Spam." Wired. June 03, 2017. Accessed July 17, 2018. https://www.wired.com/2015/07/google-says-ai-catches-99-9-percent-gmail-spam/.

up with over 50 spam emails each day. He also said, “What is even worse was that sometimes we lose genuine client emails amidst the spam emails and the worst situations occur when some of our genuine emails are classified as spam. The problem seriously affects our business.”[7]

Therefore, it can be concluded that just the presence of a spam filter is not adequate. But, there is a need for a spam filter that is accurate yet feasible at the same time.

This leads us to consider other possible algorithms to identify spam email messages. While researching for these, I came across many techniques of spam email classifiers like Artificial Neural Nets and Support Vector Machines. However, these were complicated methods that were already proven to be highly efficient and had been researched and worked on extensively. This wouldn’t give me the freedom and flexibility to experiment with these methods the way I wanted to.

Then I came across the Naïve Bayes Classifier which was a comparatively simpler machine learning technique that presented the potential to be altered and experimented with. There wasn’t a lot of research done on it as a spam email classifier and its efficiency as a classifier was variable depending on the problem it was used for. So, I found it fit to experiment with and optimize it to the problem of spam email classification. The central idea being higher accuracy, I wanted to study how the accuracy of the classifier could be maximized.

Hence, my research question:

“**How can the accuracy of a Naïve Bayes spam email classifier be maximized?**”

[7] Appendix 1

# BACKGROUND INFORMATION

## THE NAÏVE BAYES CLASSIFIER

The Naïve Bayes classifier is a method of supervised machine learning based on Bayes' Theorem of probability.

First, let us understand what supervised machine learning is. Supervised machine learning is "where there are input variables and a dependent output and an algorithm is used to learn the mapping function from the input variables to the output. 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."[8] The advantage gained from such a method is that the classifier needs to be trained only once after which it can function independently.

Now, let us understand what is the Bayes' Theorem. The Bayes' Theorem is a mathematical formula which determines conditional probability of any event based on prior knowledge of the event.

The Bayes' theorem states:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}, \qquad \text{where:}$$

- P(A) is the probability of event A
- P(B) is the probability of event B
- P(A|B) is the probability of event A given that B is a certain (true) event that occurs
- P(B|A) is the probability of event B given that A is a certain (true) event that occurs

[8] "Supervised and Unsupervised Machine Learning Algorithms." Machine Learning Mastery. September 22, 2016. Accessed July 16, 2018. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

Let us better understand this with the help of an example. Let's say we want to calculate the probability of an individual living in the United States given that he is of Asian Origin. Let the probability of picking a random individual that lives in the United States be P(A). Let the probability of picking a random individual that's Asian in origin be P(B). Therefore, P(B|A) is the probability of picking an individual that is of Asian origin given that he lives in the United States.

We are told that:

P(A) = 0.043[9]

P(B) = 0.603[10]

P(B|A) = 0.056[11]

Now, Applying Bayes' theorem we can calculate the probability of picking an individual that lives in the United States given that the individual is of Asian Origin, i.e. P(A|B).

$$P(A|B) = \frac{0.056 \times 0.043}{0.603}$$

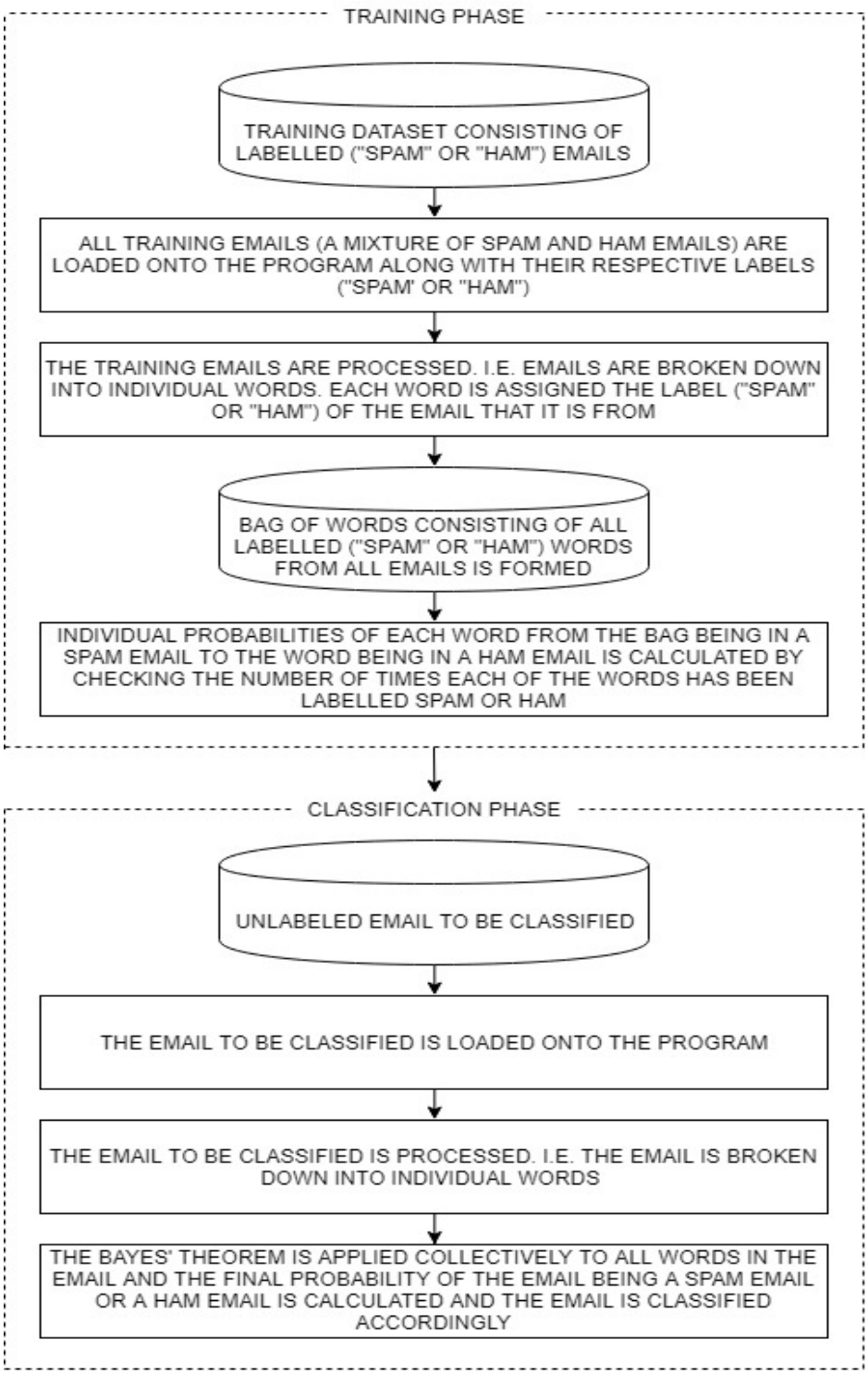
Therefore, P(A|B) = 0.004.

The Naïve Bayesian classifier uses the Bayes' Theorem to calculate probability of an object falling into a particular category based on the probability of its individual attributes.

[9] "U.S. Population (LIVE)." Philippines Population (2018) - Worldometers. Accessed July 15, 2018. http://www.worldometers.info/world-population/us-population/.
[10] "Demographics of the World." Wikipedia. July 15, 2018. Accessed July 15, 2018. https://en.wikipedia.org/wiki/Demographics_of_the_world.
[11] "Demographics of Asian Americans." Wikipedia. July 15, 2018. Accessed July 15, 2018. https://en.wikipedia.org/wiki/Demographics_of_Asian_Americans.

## THE NAÏVE BAYES SPAM EMAIL CLASSIFIER

The steps involved in the Naïve Bayes spam email classifier can be broadly classified into 2 phases: Training and Classification. The steps involved in both these phases have been outlined by the flowchart made below:



# EXPERIMENTATION

Now let us experiment and find out how different aspects of the Naïve Bayes spam email classifier can be targeted and modified to maximise its accuracy.

For experimentation, I will be using the Naïve Bayes spam email classifier that I programmed myself using Python with the help of Python libraries, mainly the Natural Language Toolkit (NLTK) which contains functions that help with natural language processing. To train my algorithm and subsequently test its accuracy, I will be using the publicly available Enron Email Corpus which is a large database of over 600,000 labelled ("spam" or "ham") emails collected from the inboxes of 158 employees of the Enron Corporation after the expose of the Enron scandal in 2001.[12]

The Enron corpus contains two subfolders labelled "spam" and "ham" which contain spam and ham mails from various employees' inboxes respectively. Each mail is stored as a separate text file (.txt). Sample spam and ham emails from the corpus are shown below:

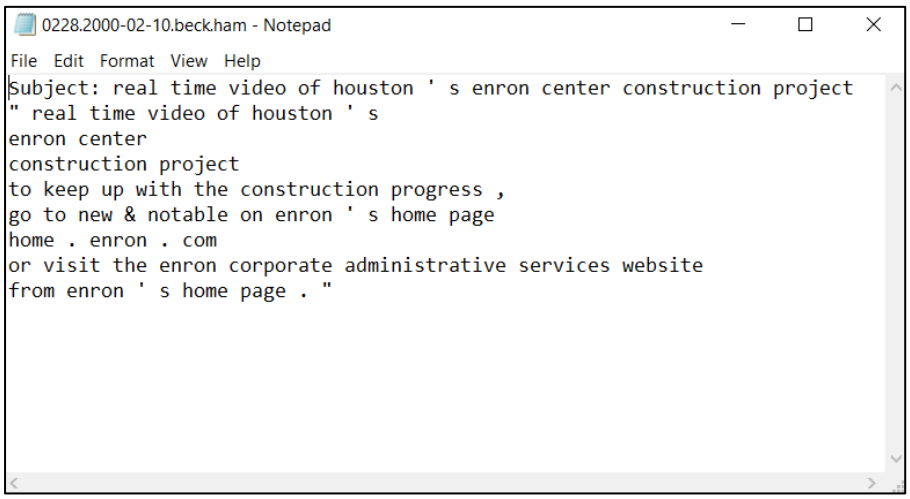



*Figure 1: A sample ham email from the Enron Corpus*

[12] Klimt, Bryan, and Yiming Yang. "The Enron Corpus: A New Dataset for Email Classification Research." Machine Learning: ECML 2004 Lecture Notes in Computer Science, 2004, 217-26. doi:10.1007/978-3-540-30115-8_22.

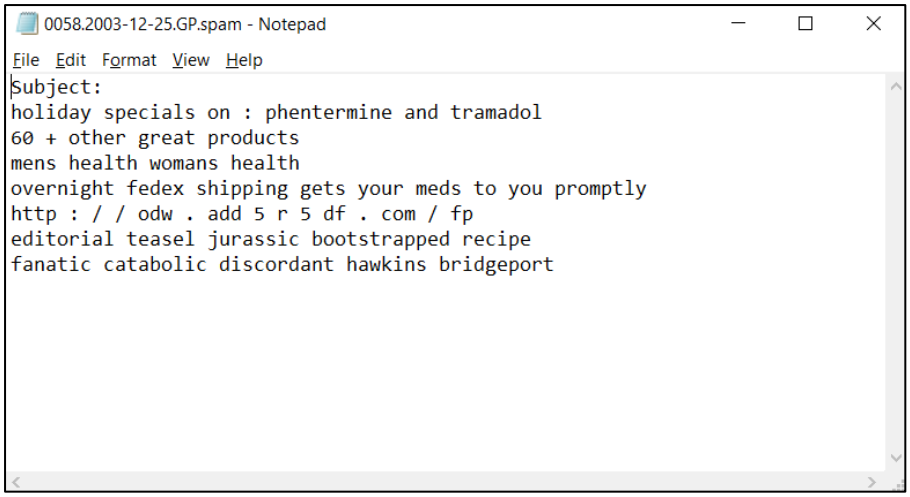



*Figure 2: A sample spam email from the Enron Corpus*

Working with the whole of such a large dataset of over 600,00 emails did not seem feasible because the processing power at my disposal was limited and continually running experiments would take up a lot of time. Therefore, I decided to limit the amount of data (emails) I use for experimentation. Thus, for each experiment, I selected an equal number of spam and ham emails to create my own unbiased training and testing datasets.

## EXPERIMENT 1: VARYING SIZE OF TRAINING DATASET

The first experiment I conducted was to test how the accuracy of the classifier responded to **changes in size of the training dataset**.

For this, I prepared 5 different training datasets of 1200, 1500, 1800, 2100 and 2400 emails each. Each of these sets contained equal number of spam and ham emails from the Enron Corpus. To test the accuracy, I created a testing dataset of 600 emails consisting of 300 spam emails and 300 ham emails collected from the Enron Corpus.

Then I ran the program 5 different times using the Naïve Bayes classifier algorithm that I programmed.[13] For the first trial, I used the 1200 email dataset to train the algorithm, for the second trial I used the 1500 email dataset and similarly for the third, fourth and fifth trails I used the 1800, 2100 and 2400 email datasets respectively. In all trials the testing dataset consisting of 600 emails was kept constant. I recorded the accuracy that was returned each time the program was run.

A sample screenshot of the results returned by one run of the classifier is explained below (all results of all further experiments are also returned in a similar format):

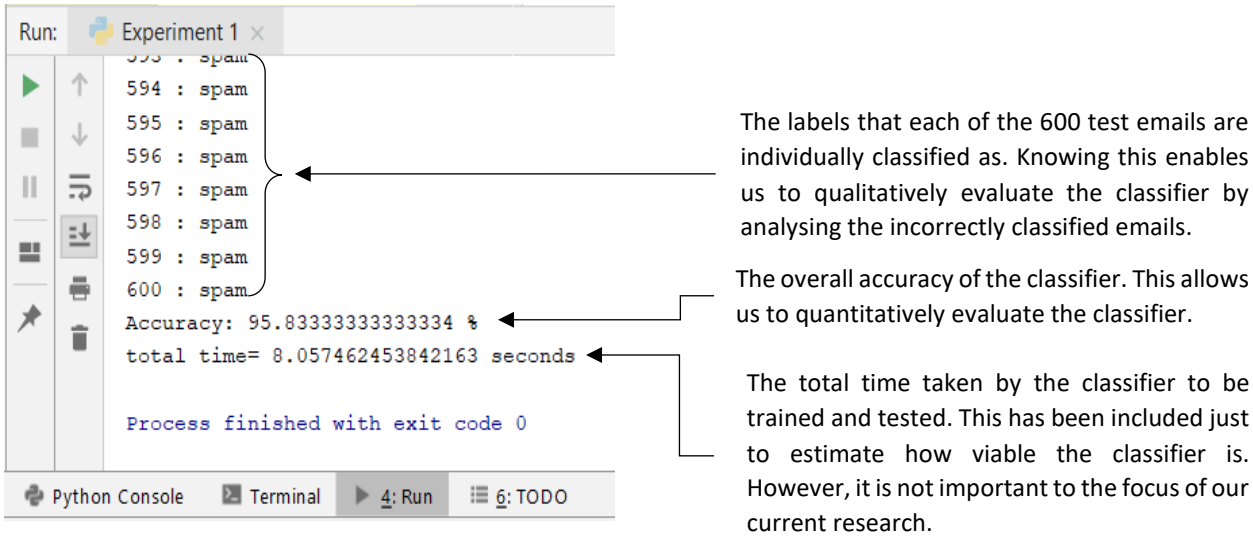


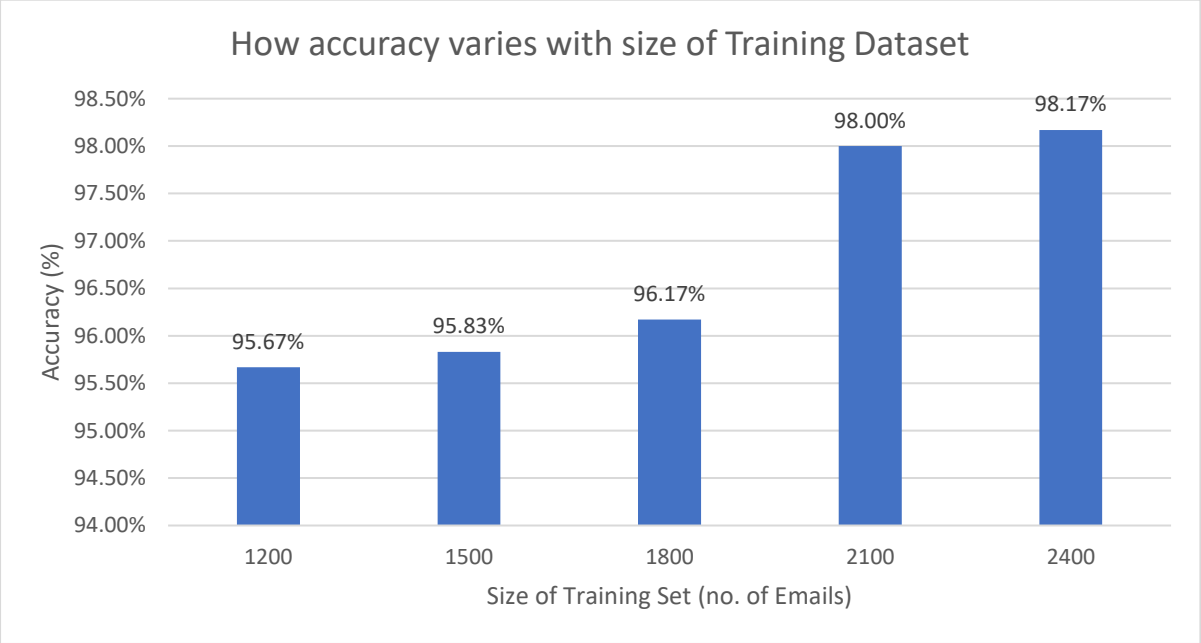

*Figure 3: Sample screenshot of the results of an experiment*

The results of Experiment 1 have been summarized in the table below:

| Test No. | Size of Training Dataset | Size of Testing Dataset | Accuracy |
|---|---|---|---|
| 1 | 1200 emails | 600 emails | 95.67% |
| 2 | 1500 emails | 600 emails | 95.83% |
| 3 | 1800 emails | 600 emails | 96.17% |
| 4 | 2100 emails | 600 emails | 98.00% |
| 5 | 2400 emails | 600 emails | 98.17% |

[13] Appendix 2

For clearer analysis, the results of the experiment have been graphically represented by the bar-graph below:




It was observed that as the size of the training set increased, the accuracy of the classifier also increased. Therefore, it can be concluded that the accuracy of the classifier is directly proportional to the size of the training set. This is because with each email the classifier is trained with, the words of the email get a higher probability of being either spam or ham and hence become more well defined. Since the values get more absolute, the classifier can be said to be better trained and it can better estimate the probabilities of any email being ham or spam.

However, for the sake of fair further experimentation, I have chosen a training set of 1500 emails and kept the size of my testing set constant at 600 emails to better see the difference of modifying other features of the classifier on its accuracy.

## EXPERIMENT 2: VARYING LANGUAGE PROCESSING

Many times, spam email senders use various techniques to trick content-based spam filters and reach the user inbox. This can include using varying spellings of common spam words like saying “cocane” instead of “cocaine” so that the spam filter does not recognize the usage of a common spam word “cocaine” in the email. The spam sender may even emphasize words by adding extra letters to them so that the word with extra letters is not identified as a spam word by the classifier. For example, saying “sexxx” instead of “sex”. Such techniques may cause the filter to classify a spam email incorrectly and therefore greatly hamper the efficiency of a spam email classifier. Thus, the spam filter is required to process the emails in such a way so as to tackle such techniques used by spammers.

So, the second experiment I conducted was to see how the accuracy of the classifier responded to **changes in the way I processed each email; i.e. how I processed language.**

To understand what the different ways of processing the language in the emails, I decided to consult a Machine Learning expert from Carnegie Mellon University- Sparsh Chandra. Through an interview with him[14], I learned the various functions that can be applied to words of an email to yield a more definitive probability of it being a spam email or ham email. After considering the complexity and feasibility of the functions, I have chosen to process the language of the email by ignoring stop-words, removing extra letters and applying a spell-check feature. Each of these have functions have been further explained below.

Stop-words are the most commonly words used in a language such as “is”, “an” and “the”. We may ignore stop-words because they occur equally frequently in spam emails as well as

[14] Appendix 3

ham emails and hence exclude them when calculating the probability of the email being ham or spam. For example, “A lion ate him up” with stop-words removed would be “lion ate”. There is no definitive list of stop-words in English. However, the NLTK library has a list of common English stop-words which I have employed in my experimentation.

Removing extra letters from a word helps catch spam words where extra letters have been added intentionally with the sole purpose of adding emphasis and surpassing spam filters. It is known that no word in the English dictionary has more than 2 same consecutive letters. Therefore, all words with more than 2 repeating letters anywhere throughout the word, must have the extra letters removed. For example, if an email contains the word “freeeeeee”, a spam filter may not be trained to recognize this word. However, cutting down extra letters will modify the word to “free” to ensure that the probability of “freeeeeee” being in a spam email or ham email is the same as that of the word “free”.

Checking spellings of words in an email and correcting them is another useful way of tackling spam email senders who purposely use wrong spellings for certain words to deceive the filter. For example, if an email contains the words “girlz”, chances are that the filter is not trained to recognize this word. However, a spell check function will automatically change it to “girls” and the probability of it being in a spam email or a ham email is then same as that of the word “girls”. For my experimentation, I will use the inbuilt spell-check feature of the NLTK.

I tried all possible combinations of applying these 3 functions to the classifier. The table on the next page represents the trial number and the functions applied to the classifier in the trial and the corresponding accuracy returned by the classifier.

| Trial Number | Ignoring stop words | Removing extra letters | Spell check | Accuracy yielded |
|---|---|---|---|---|
| 1 | ✗ | ✗ | ✗ | 95.83% |
| 2 | ✓ | ✗ | ✗ | 95.53% |
| 3 | ✗ | ✓ | ✗ | 96% |
| 4 | ✗ | ✗ | ✓ | 95.53% |
| 5 | ✓ | ✓ | ✗ | 95.67% |
| 6 | ✓ | ✗ | ✓ | 95.67% |
| 7 | ✗ | ✓ | ✓ | 95.67% |
| 8 | ✓ | ✓ | ✓ | 95.67% |

The accuracy returned by all these experiments are represented by the Venn Diagram below:

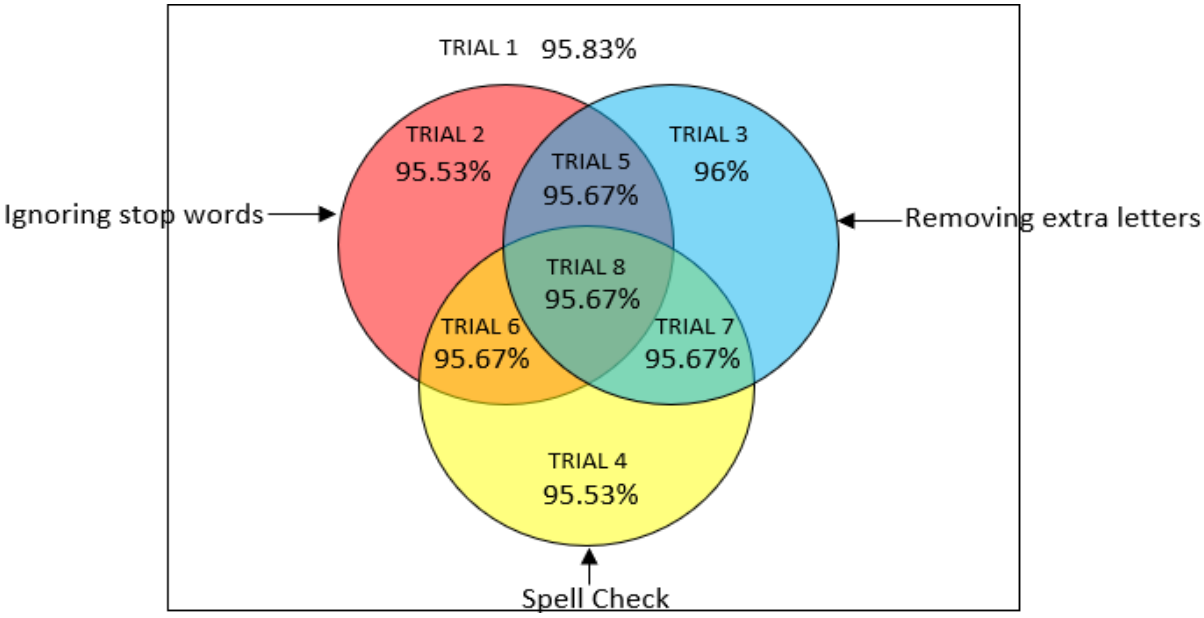



*Figure 4: Venn Diagram representing accuracies yielded from trials of Experiment 2*

As evident from the results of the experiment, it is observed that Trial 3 (only removing extra letters) yields the highest accuracy (96%). Surprisingly, all other functions in other combinations decrease the accuracy of the classifier. This may be because spam and ham

emails may be characterized with usage of stop words or incorrect spellings to different degrees.

For example, as evident from the sample spam and ham emails from the corpus shown in Figure 1 and Figure 2, spam emails generally contain fewer stop words as compared to ham emails which means they have a greater probability of occurring in spam emails than in ham emails. Therefore, excluding stop words from our calculations may lead to diluting the probability of an email being ham or spam and hence reduce the overall accuracy of the classifier.

On further analysing this, I even found that certain spam emails even follow similar trends of wrong spellings. For example, the 2 spam emails from the corpus shown below (Figure 3 and Figure 4) consistently mis-spell words like money as “mony”, guarantee as “garnte” and shipped as “shpped”.

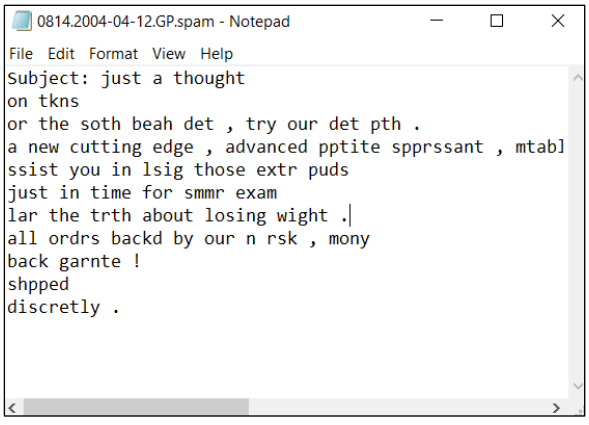



*Figure 5: A spam email from the Corpus*

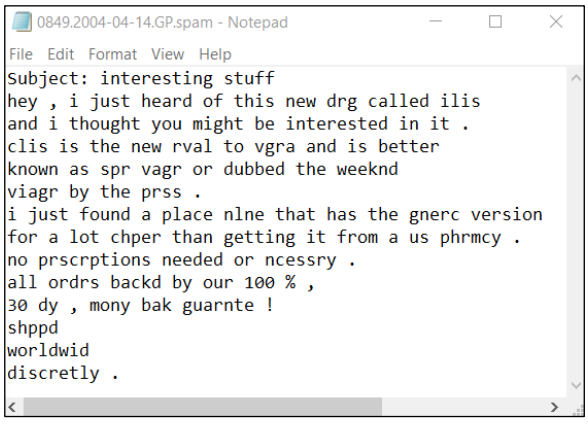



*Figure 6: A spam email from the Corpus*

In such cases, applying the spell check function would mean mixing the probability of the words like “mony” with “money” even though the incorrect spelling would have a greater probability of being in a spam email compared to a ham email.

So, for further experimentation, I decided to only removing extra letters from the emails when processing them.

## EXPERIMENT 3: VARYING THRESHOLD VALUES

Not all spam emails may be about advertising ingenuine products or phishing scams. And at the same time, not all ham emails may be just about work or formal communications. Some spam emails may sound like genuine emails conveying seemingly regular messages, whereas on the other hand many ham emails may seem like spam. On many occasions the classifier may classify some spam emails as ham emails known as false negatives, or classify some ham emails as spam, known as false positives.[15]

Classifying ham emails as spam is worse because it implies loss of genuine emails that may contain important information and have severe consequences. Thankfully, we can control the number of false positives and false negatives by varying the "threshold value" of the classifier. The threshold value refers to the minimum probability of an email being a spam email for it to be classified as a spam email. For example, if the threshold value of the classifier is set as 0.8, only emails whose probability of being a spam email is greater than or equal to 0.8 will be classified as spam emails and the rest will be classified as ham.

For all experiments up till now the threshold value has been 0.5. If we increase the threshold value, it is expected that the number of false positives will increase while false negatives will decrease and vice-versa if we decrease the threshold value. Therefore, for my third experiment, I decided to see how the accuracy of the classifier responded to **changes in the threshold value of the classifier**.

[15] "False Positives and False Negatives." Wikipedia. October 23, 2018. Accessed October 28, 2018. https://en.wikipedia.org/wiki/False_positives_and_false_negatives.

So, I varied the threshold value of the classifier from values between 0.1 to 0.9 and track the number of false positives, false negatives, and the corresponding accuracy. The results of this experiment are summarized by the table below:

| Trial Number | Threshold value | False Positives | False Negatives | Accuracy yielded |
|---|---|---|---|---|
| 1 | 0.1 | 26 | 0 | 95.67% |
| 2 | 0.2 | 24 | 0 | 96% |
| 3 | 0.3 | 23 | 1 | 96% |
| 4 | 0.4 | 22 | 1 | 96.17% |
| 5 | 0.5 | 22 | 2 | 96% |
| 6 | 0.6 | 22 | 2 | 96% |
| 7 | 0.7 | 22 | 2 | 96% |
| 8 | 0.8 | 21 | 2 | 96.17% |
| 9 | 0.9 | 19 | 4 | 96.17% |

As the threshold value is increased, any email will require a higher probability of being a spam email to be classified as spam. At the same time, ham emails that show some spam email characteristics are protected against being classified as spam. Therefore, as the threshold value is increased, the number of false negatives (spam emails classified as ham emails) increases, whereas the number of false positives (ham emails classified as spam emails) decreases. While the optimum accuracy, 96.17%, is returned by 3 threshold values, 0.4, 0.8 and 0.9, at 0.9 the false positives are minimum and as discussed before, having false negatives is preferable over false negatives.

Therefore, the optimum threshold value is 0.9 since it yields maximum accuracy of 96.17% while keeping false negatives as low as possible.

## LIMITATIONS:

The experiments conducted by me faced several limitations. Most of these proved to be consequences of the limited computational power at my disposal. The 2 main limitations faced during experimentation, besides the inability to use a larger dataset as mentioned earlier, are the inability to use N-grams and the inability to use media and image classification in the classifier.

As discussed earlier, the Naïve Bayes classifier makes the "naïve" assumption that all attributes (words) of the email are independent of each other. This means that the classifier treats all words in any email in the same manner irrespective of the context. For example, let's say that Email 1 reads "This email is not spam" and Email 2 reads "This email is spam". When calculating the probabilities of Email 1 and Email 2 being spam or ham, the classifier will consider the probability of the word "spam" being in spam emails and ham emails without considering the fact that it is preceded by the word "not" in Email 1.

To take "phrases" and adjacent words into consideration, we can use N-grams. "N-grams are consecutive sequences of tokens instead of singular tokens, where the tokens are either words or characters."[16] For example, A bigram would take every 2 adjacent tokens or words into consideration and calculate the probability of the exact phrase formed being in a spam email or ham email. For example, if we use the bigram approach to the mentioned example, the classifier would calculate the probability "not spam" in the first email and "is spam" in the second email separately and therefore would not confuse the use of the word "spam" in both emails.

[16] Using Naive Bayes and N-Gram for Document Classification. (n.d.). Retrieved from http://kth.diva-portal.org/smash/get/diva2:839705/FULLTEXT01.pdf

However, using N-grams increases the number of processes and phrases to keep track of, significantly increasing the complexity of the code and exponentially increasing the processing time.[17] Therefore, N-grams could not be included in my research.

Spam emails very often contain only media files with no textual context. This is evident from a commonly wrongly classified email across all experiments as shown below:

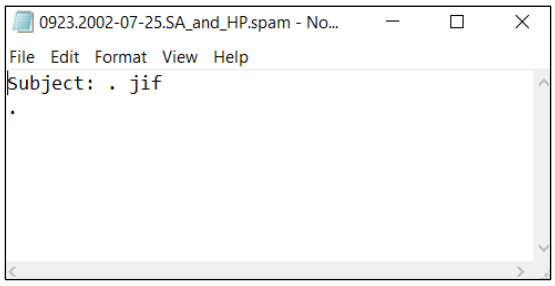



*Figure 7: A commonly wrongly classified spam email from the Enron Corpus*

This email suggests an attached media file that may have contained spam material. However, not only did the Enron Corpus not contain media files, but also including media in the classifier would be impossible considering an entirely new classifier would have to be made to classify the media which I found to be very complex and beyond the scope of my research.

# CONCLUSION

The accuracy of the Naïve Bayes spam email classifier can be significantly increased by adjusting various attributes of the classifier.

Experiment 1 proved that the accuracy of the classifier is directly proportional to the size of the dataset used to train it, so greater the amount of labelled emails that are used to train the classifier, the greater is its accuracy. Experiment 2 revealed that removing extra letters

[17] By. "NGrams and Edge NGrams - Computational Complexity." Jakub Staš. October 24, 2017. Accessed October 28, 2018. https://jakubstas.com/ngrams-edge-ngrams-computational-complexity/#.W9WRApMzZPY.

from words further improved the classifier's accuracy. And Experiment 3 suggested that 0.9 is the optimum threshold value for an email to be classified as a spam email by the classifier because it yields the highest accuracy while keeping the number of false negatives as low as possible.

I decided to find the maximum accuracy I could achieve using my classifier. So, combining the findings of all experiments by using all 600,000 emails of the Enron Corpus to train the classifier, removing extra letters from words of the email when processing it and using a threshold value of 0.9, I ran the classifier one last time[18] and achieved a pinnacle accuracy of **98.5%**!

The above documented research suggests that the Naïve Bayes method of Machine Learning has potential to yield a very high accuracy for spam email classification. It even possesses scope for modifying the classifier to suit the needs of the user. For example, if a user wants a spam-free inbox and doesn't mind some of his ham emails being directed to the spam folder, he can choose to set a low threshold value for the classifier. Conversely, if he doesn't mind a few spam emails in his inbox but cannot afford to lose any ham emails, he can set the value a high threshold value for his classifier.

Of course, the accuracy of the classifier can be further enhanced by using larger training datasets and the use of N-grams provided enough computational ability. The scope of the research can also be expanded by including media files when classifying emails by building a classifier specific to the nature of the media and further incorporating it into the actual Naïve Bayes classifier.

[18] Appendix 4

# BIBLIOGRAPHY


1. "15 Outrageous Email Spam Statistics That Still Ring True in 2018." RSS. Accessed July 15, 2018. https://www.propellercrm.com/blog/email-spam-statistics.
2. "AI Final 1." Accessed October 28, 2018. https://www.csie.ntu.edu.tw/~r98922004/doc/AI_final.pdf.
3. "Build a Spam Filter." Python For Engineers. Accessed October 28, 2018. https://www.pythonforengineers.com/build-a-spam-filter/.
4. "Enron Corpus." Wikipedia. July 03, 2018. Accessed July 19, 2018. https://en.wikipedia.org/wiki/Enron_Corpus.
5. "KDnuggets." KDnuggets Analytics Big Data Data Mining and Data Science. Accessed October 28, 2018. https://www.kdnuggets.com/2017/03/email-spam-filtering-an-implementation-with-python-and-scikit-learn.html.
6. "Machine Learning Methods for Spam E-Mail Classification." Accessed October 28, 2018. https://www.researchgate.net/publication/50211017_Machine_Learning_Methods_for_Spam_E-Mail_Classification.
7. "Malware Definition – What Is It and How to Remove It." Malwarebytes. Accessed December 08, 2018. https://www.malwarebytes.com/malware/.
8. "Naive Bayes Spam Filtering." Wikipedia. October 14, 2018. Accessed October 28, 2018. https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering.
9. "Natural Language Toolkit." Wikipedia. August 15, 2018. Accessed October 28, 2018. https://en.wikipedia.org/wiki/Natural_Language_Toolkit.
10. "Natural Language Toolkit¶." Natural Language Toolkit - NLTK 3.3 Documentation. Accessed October 28, 2018. https://www.nltk.org/.

11. "Ways to Improve the Accuracy of a Naive Bayes Classifier?" Stack Overflow. Accessed October 28, 2018. https://stackoverflow.com/questions/3473612/ways-to-improve-the-accuracy-of-a-naive-bayes-classifier.
12. 1451847111494285. "How To Build a Simple Spam-Detecting Machine Learning Classifier." Hacker Noon. April 01, 2017. Accessed October 28, 2018. https://hackernoon.com/how-to-build-a-simple-spam-detecting-machine-learning-classifier-4471fe6b816e.
13. By. "NGrams and Edge NGrams - Computational Complexity." Jakub Staš. October 24, 2017. Accessed October 28, 2018. https://jakubstas.com/ngrams-edge-ngrams-computational-complexity/#.W9WRApMzZPY.
14. Deshpande, Bala. "3 Challenges with Naive Bayes Classifiers and How to Overcome." Analytics Made Accessible: For Small and Medium Business and beyond. Accessed October 28, 2018. http://www.simafore.com/blog/3-challenges-with-naive-bayes-classifiers-and-how-to-overcome.
15. Klimt, Bryan, and Yiming Yang. "The Enron Corpus: A New Dataset for Email Classification Research." Machine Learning: ECML 2004 Lecture Notes in Computer Science, 2004, 217-26. doi:10.1007/978-3-540-30115-8_22.
16. Using Naive Bayes and N-Gram for Document Classification. (n.d.). Retrieved from http://kth.diva-portal.org/smash/get/diva2:839705/FULLTEXT01.pdf.
17. Youn, Seongwook, and Dennis McLeod. "A Comparative Study for Email Classification." SpringerLink. January 01, 1970. Accessed October 28, 2018. https://link.springer.com/chapter/10.1007/978-1-4020-6264-3_67.

# APPENDICES:

## APPENDIX 1: INTERVIEW WITH MR. PRATIK SHAH:

Produced below is the transcript of my interview with Mr. Pratik Shah.

Me (M): Hello Mr. Pratik. Thank you for meeting with me.

Mr. Shah (P): Hello. Sure, no problem.

M: So, I am here today because I want to understand some problems that your company faces because of spam emails. First off, could you please tell me what you do and what is it that your company does?

P: Sure. My company, Shivji Valji Tarpaulins Pvt. Ltd. is basically a multipurpose shed provider. We build sheds and structures for factories, construction sites, monsoon structures, etc. I am the managing co-director of the company.

M: Oh okay. That sounds interesting. How many employees do you have in your company? Is email an important method of communication for your company?

P: We have a team of 15 employees. And yes, email is a very popular and important method of communication in our company. All external communications ranging from client requests for sheds to complaints against shed structures to payment quotations, all of these are done via email.

M: Hm, alright. Since email is such an important method of communication, does your company have its own email domain and a distinct email service provider?

P: Yes, we do.

M: Alright. What is the most prominent issue you face with your emails?

P: SPAM EMAILS! Spam emails are a big nuisance. Spam emails reduce our employee's productivity I the form of distractive content. To protect our computers from malware in emails, we have to invest in anti-malware software for all our employees. We receive spam email in such large quantities that it actually affects our email storage space and sometimes our email networks get jammed because of these.

M: That's quite true. Spam emails are one of the most serious problems modern businesses face. But doesn't your email service provider provide a spam email filter? So that your inboxes are clean and unaffected?

P: Our email service provider, whose identity we cannot disclose for because that is confidential, does have a spam email filter in place for us. However, it is utterly useless! On regular days, our employees' inboxes fill up with over 50 spam emails! The classifier that is in place for us is highly inefficient. What is even worse is that sometimes we lose genuine client emails amidst the spam emails. This problem seriously affects our business.

M: Oh, that must be really disappointing. But I have understood your problem. I am currently researching a method to create more efficient spam email filters. Thank you for sharing your problem with me. I will consider this in my research.

P: Okay. Thank you.

## APPENDIX 2: BASIC NAÏVE BAYES ALGORITHM

Produced below is the Naïve Bayes Algorithm that I have used for my experiment. However, this is just the basic code. The code for each run of the experiment is unique and differs slightly based on the attribute being tested.

```
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import stopwords
```

```
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from autocorrect import spell
import re
import os
import random
import time

def main ():
    start_time=time.time()
    rootdir="C:\\Training dataset 1200"

    ham_train_list = [] # to store the training ham emails
    spam_train_list = [] # to store the training spam emails
    ham_test_list = [] # to store the testing ham emails
    spam_test_list = [] # to store the testing spam emails

    checkpoint_time1=time.time()
    print("time elapsed=", (checkpoint_time1-start_time))
    print("training in progress...")
    print("processing training files...")

    for directories, subdirs, files in os.walk(rootdir):

        if (os.path.split(directories)[1]=="ham"):
            for file in files:
                with open(os.path.join(directories, file),
encoding="latin-1") as f:
                    data = f.read()
                    words = word_tokenize(data)

ham_train_list.append((create_word_features(words), "ham"))
            print("file end time check")
            checkpoint_time2=time.time()
            print("time elapsed=", (checkpoint_time2-start_time))

        if (os.path.split(directories)[1]=="spam"):
            for file in files:
                with open(os.path.join(directories, file),
encoding="latin-1") as f:
                    data=f.read()
                    words = word_tokenize(data)

spam_train_list.append((create_word_features(words), "spam"))
            print("file end time check")
            checkpoint_time3=time.time()
            print("time elapsed=", (checkpoint_time3-start_time))

    print("processing testing files...")
    checkpoint_time4=time.time()
    print("time elapsed=", (checkpoint_time4-start_time))

    rootdir="C:\\Testing Dataset 600"

    for directories, subdirs, files in os.walk(rootdir):
```

```
            if (os.path.split(directories)[1]=="ham"):
                for file in files:
                    with open(os.path.join(directories, file),
encoding="latin-1") as f:
                        data = f.read()
                        words = word_tokenize(data)

ham_test_list.append((create_word_features(words)))
                print("file end time check")
                checkpoint_time2=time.time()
                print("time elapsed=", (checkpoint_time2-start_time))

            if (os.path.split(directories)[1]=="spam"):
                for file in files:
                    with open(os.path.join(directories, file),
encoding="latin-1") as f:
                        data=f.read()
                        words = word_tokenize(data)

spam_test_list.append((create_word_features(words)))
                print("file end time check")
                checkpoint_time3=time.time()
                print("time elapsed=", (checkpoint_time3-start_time))

    training_set=ham_train_list+spam_train_list # to store the
complete training dataset
    testing_set=ham_test_list+spam_test_list # to store complete
training dataset


    print("checkpoint 3")
    checkpoint_time5=time.time()
    print("time elapsed=", (checkpoint_time3-start_time))

    classifier = NaiveBayesClassifier.train(training_set)

    x=1
    fp=0
    fn=0
    for mail in testing_set:
        dist = classifier.prob_classify(mail)
        spam_prob=dist.prob("spam")
        if (spam_prob>0.5):
            print(x,": spam")
            if(x<=300):
                fp=fp+1

        else:
            print(x,": ham")
            if(x>300):
                fn=fn+1
        x+=1
    print("Accuracy:", (((600-(fp+fn))/600)*100), "%")
    print("total time=", time.time()-start_time, "seconds")

def create_word_features(words):
```

```
    my_dict = dict([(word), True) for word in words])
    return my_dict

if __name__ == "__main__":
    print("classifier is running...")
    main()
```

## APPENDIX 3: INTERVIEW WITH SPARSH CHANDRA

Me (M): Hello Sparsh. Thank you for agreeing to converse with me.

Sparsh (S): Hi. No issues.

M: I know your time is very valuable. So, I will keep this brief. Are you aware of the Naïve Bayes Method of Machine Learning.

S: Yes, I am aware. I myself have worked on it extensively.

M: Okay, perfect. What about its application in language processing? How effective is it?

S: The Naïve Bayes method of Machine Learning is extremely effective for language processing. It has a lot of potential to be altered and experimented with to suit the needs of any application.

M: Okay, that's perfect! So, currently, I am working on its application in Spam Email Classification. I want to study how varying the type of language processing can affect the accuracy of my classifier. Can you suggest how I should go ahead with this?

S: Oh, wonderful! Yes, sure! Okay so I suggest you try the following 3 features. Research them thoroughly and try different combinations. Try ignoring stop-words, removing extra letters from words and applying a spell-check feature. I will not give you more details because I do not want to influence your research. But these are the basic functions you should try.

M: Oh, okay. That's really helpful. I will definitely research and try these. Thank you so much for your valuable time.

S: Thank you. And all the best for your research.

## APPENDIX 4: FINAL NAÏVE BAYES ALGORITHM

Produced below is the final code of my Naïve Bayes Classifier. This has been edited to include the whole Enron Corpus (600,000 emails) as the training dataset, remove extra letters from all words when processing them and have a threshold value of 0.9. This classifier returned an accuracy of 98.5%.

```
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from autocorrect import spell
import re
import os
import random
import time

def main ():
    start_time=time.time()
    rootdir="C:\\Enron Spam"

    ham_train_list = [] # to store the training ham emails
    spam_train_list = [] # to store the training spam emails
    ham_test_list = [] # to store the testing ham emails
    spam_test_list = [] # to store the testing spam emails

    checkpoint_time1=time.time()
    print("time elapsed=", (checkpoint_time1-start_time))
    print("training in progress...")
    print("processing training files...")

    for directories, subdirs, files in os.walk(rootdir):

        if (os.path.split(directories)[1]=="ham"):
            for file in files:
                with open(os.path.join(directories, file),
encoding="latin-1") as f:
                    data = f.read()
                    words = word_tokenize(data)

ham_train_list.append((create_word_features(words), "ham"))
```

```
            print("file end time check")
            checkpoint_time2=time.time()
            print("time elapsed=", (checkpoint_time2-start_time))

        if (os.path.split(directories)[1]=="spam"):
            for file in files:
                with open(os.path.join(directories, file),
encoding="latin-1") as f:
                    data=f.read()
                    words = word_tokenize(data)

spam_train_list.append((create_word_features(words), "spam"))
            print("file end time check")
            checkpoint_time3=time.time()
            print("time elapsed=", (checkpoint_time3-start_time))

    print("processing testing files...")
    checkpoint_time4=time.time()
    print("time elapsed=", (checkpoint_time4-start_time))

    rootdir="C:\\Testing Dataset 600"

    for directories, subdirs, files in os.walk(rootdir):

            if (os.path.split(directories)[1]=="ham"):
                for file in files:
                    with open(os.path.join(directories, file),
encoding="latin-1") as f:
                        data = f.read()
                        words = word_tokenize(data)

ham_test_list.append((create_word_features(words)))
                print("file end time check")
                checkpoint_time2=time.time()
                print("time elapsed=", (checkpoint_time2-start_time))

            if (os.path.split(directories)[1]=="spam"):
                for file in files:
                    with open(os.path.join(directories, file),
encoding="latin-1") as f:
                        data=f.read()
                        words = word_tokenize(data)

spam_test_list.append((create_word_features(words)))
                print("file end time check")
                checkpoint_time3=time.time()
                print("time elapsed=", (checkpoint_time3-start_time))

    training_set=ham_train_list+spam_train_list # to store the
complete training dataset
    testing_set=ham_test_list+spam_test_list # to store complete
training dataset

    print("checkpoint 3")
    checkpoint_time5=time.time()
    print("time elapsed=", (checkpoint_time3-start_time))
```

```
    classifier = NaiveBayesClassifier.train(training_set)

    x=1
    fp=0
    fn=0
    for mail in testing_set:
        dist = classifier.prob_classify(mail)
        spam_prob=dist.prob("spam")
        if (spam_prob>0.9):
            print(x,": spam")
            if(x<=300):
                fp=fp+1

        else:
            print(x,": ham")
            if(x>300):
                fn=fn+1
        x+=1

    print("Accuracy:", (((600-(fp+fn))/600)*100), "%")

    print("total time=", time.time()-start_time, "seconds")

def create_word_features(words):
    my_dict = dict([(re.sub(r'(.)\1+', r'\1\1', word)), True) for
word in words])
    return my_dict

if __name__ == "__main__":
    print("classifier is running...")
    main()
```