CSCI4511W Project: Naïve Bayesian Spam Filters with n-grams

Abstract

Unsolicited emails (spam) are being sent at an ever increasing rate. In an effort to block these disruptive emails, spam filters have been developed to automatically block spam emails. Naive Bayesian filters are one of the more commonly used types of filters used to block spam in commercial spam email filters. Naive Bayesian filters come in a variety of different forms that each seek to distinguish legitimate emails from spam emails. In our paper, we explore three different types of Naives Bayes classifiers - Multinomial Naive Bayes with Term Frequency Attributes, Multinomial Naive Bayes with Boolean Attributes, and Multi-variate Bernoulli Naive Bayes. We also examine alterations to the Naive Bayes filters such as the usage of multiple-word strings as tokens called n-grams, rather than only single words. We ran these Naive Bayes filters on emails gathered from the mailboxes of employees who worked at Enron, which were released following the Enron investigation in 2001. We compare the results of filtering using the different types of Naive Bayes filters as well as using n-grams. The metric we used to analyze the performance of the filters is in terms of false positive (classifying legitimate emails as spam) and false negatives (classifying spam emails as legitimate). Finally we discuss the outcome of our tests and what other work could be done to implement a higher quality filter.

1 Introduction

As the number of email users increases, the need for spam filters becomes a very important part of today's online world. Many companies today use emails to allow their employees to effectively exchange messages with colleagues or to reach out to potential customers. Emails are not only limited to commercial purposes though, as many people enjoy using personal email accounts to communicate at any distance with friends and family. Unfortunately, this has also given an efficient medium for people to bombard email users with scam emails and undesired advertising emails. These annoying spam emails are sent out to many email users in the world with little regard for who is actually receiving the email. In a study conducted by Kaspersky Lab ZAO in Q3 2015, spam accounted for 54.2% of all email traffic [18]. It can be seen why with the rate at which spam propagates through the Internet, that there has been a growing interest in reducing the amount of spam email that an email user would receive.

While spam emails are obnoxious and clutter to the average email user's email inbox, they can also be costly for businesses. In 2012, this vast amount of spam was estimated to cost between 20 and 50 billion dollars annually to American firms and consumers [15]. This means that spam filters are not only improve the quality of an email user's inbox, but they also can help increase an employee's productivity and save the company money. Employees will have to waste the companies' time wading through pointless emails that they have no intention of following up with. Another issue besides the economic and time wasting strains that spam emails have on a company is that they can send unacceptable content for work. Some spam emails can contain explicit or offensive content such as links to pornographic websites. For these reasons, anti-spam countermeasures have been taken in a variety of different ways.

One approach that has been taken and is mainly discussed in this paper is filtering spam email based on the textual content of the email. The goal of a text based spam filter is to categorize an email as legitimate

or spam based on the message received. This is typically done by iterating through the email's body and/or subject line to gather the textual content in an email. After the information is collected from the email, one of many different algorithms is run on the data to determine likelihood of that given email to be spam.

In this paper, we focus on the algorithms and techniques for spam filtering of the Naive Bayesian variety. Naive Bayes classifiers are probabilistic classifiers that apply Bayes' theorem to separate inputs into two different categories. In our case, we seek to separate email inputs into one of two categories: ham (legitimate emails), and spam (non-legitimate emails). We work with three types of Naive Bayes classifiers which are Multinomial Naive Bayes with Term Frequency Attributes, Multinomial Naive Bayes with Boolean Attributes, and Multi-variate Bernoulli Naive Bayes. Based on these methods of classifying emails, an email user's inbox could be organized by having ham emails and spam emails divided into separate collections. Although these algorithms can fail to accurately classify emails as seen by our experiments, classifying emails in this manner can be very beneficial to an individual email user.

We conducted our experiments using the dataset provided by Metsis et al. [11]. This dataset includes emails gathered from the mailboxes of employees who worked at Enron, which were released following the Enron investigation in 2001. These emails are stored in the form of text files where both the subject line and the body of the emails are written in plain text. We trained our filters on the approximately 30000 ham and spam emails that were provided by Enron. We then ran tests on 1500 ham mails and 4500 spam mails that were removed from the original dataset. For each test we used one of the three types of Naive Bayes filters mentioned earlier.

In the following sections, we describe what other researchers have done in regards to spam filtering, which can include Naive Bayesian and other text based spam filtering as well as non-textual based methods. Then, we explain our approach and discuss our results after running three types of Naive Bayesian filters on sample ham and spam emails. Finally, we analyze our results and explore potential work for the future.

2 Related Work

According to Garcia et al. [4], approaches to countering spam can be divided into two distinct groups. The first method attempts to curb the total number of spam emails sent by either blocking or limiting access to mail servers for spammers. This can be done in a few different ways.

The most straightforward way to prevent spam is to close all open relays on the Internet and to strengthen the SMTP (Simple Mail Transfer Protocol) to block fake headers and require sender authentication. This would force spammers to send their spam through their ISP and relies on the ISPs to block those accounts. However, this approach attacks the openness of the Internet and threatens Internet privacy. This approach may not even work for very long, as spammers can simply shift to using proxies or botnets.

Another method is to use money or computation time as a price for sending emails [7]. This approach may disinsentivize spammers from sending large amounts of spam. Forcing senders of email to pay a small fee in order to send an email could make the cost of sending thousands of spam emails prohibitive, while keeping the cost reasonable for the average user. There could also be methods in place that allow whitelisting contacts so that correspondence doesn't require payment. This could be thought of as a "electronic postage" approach.

However, this approach would require many sacrifices to be made by Internet users. There could be substantial overhead costs to implement while centralizing the Internet to some degree, and success would require widespread adoption by many users and ISPs. It would likely have a large opposition, as it would require people to give up their free communication and could threaten privacy as well. Finally, to maintain privacy, some sort of anonymous electronic cash would need to be used, and widespread adoption of this system is unlikely.

The alternative form of "pricing" email would be to force the sender of emails to compute expensive functions in order to send the email. [3] For normal users, the extra time required to send an email would be nearly unnoticeable, but for spammers, the extra time added to each of the thousands or hundreds of thousands of sent emails would make the calculations prohibitively long. This would allow email to still be

free, would not require centralization, and could allow whitelisting so that senders could send to contacts who whitelisted them without "paying."

The downside to this approach is the difference in computing power between systems. In order to combat spammers with modern hardware, the computation time required to solve the "pricing function" would need to be calculated for modern machines. However, this means that normal users with old hardware may not be able to solve the function in a reasonable amount of time.

Both of these methods would require instant global adoption to be effective and to prevent people from losing correspondence if one of them does not update to the new method. Modern physical junk mail also shows that most spammers and advertisers would also still be willing to pay to send their messages regardless. All in all, these approaches seem difficult to implement and unlikely to come about without massive cooperation. Therefore, we should turn our focus toward spam filtering.

The second basic approach to countering spam is to use a spam filter. This approach attempts to detect and remove spam once it is sent by applying different filtering techniques to received messages. These techniques can be divided into two major categories: cooperative filtering and heuristic filtering.

Cooperative filtering includes a few separate approaches [7]. The first method exploits the fact that spam messages are by nature very similar. Messages marked a spam are hashed and sent to a database such as Vipul's Razor [14] or the Distributed Checksum Clearinghouse (DCC) [19]. Then, users can hash messages sent against those in these databases and delete those that match known spam messages. This system, however, can be abused by submitting hashes of legitimate messages, either deliberately or accidentally. This method can also be circumvented by adding unique tags to each individual message, giving them different hash values.

A second approach would consist of opt-in and opt-out lists, where spam generators would cooperate with users and not send spam to those on opt-out lists. However, since most spammers ignore existing opt-out lists, this approach cannot be seriously considered. Since cooperative filtering seems to be easily exploitable, we must focus on heuristic filtering.

Heuristic-based filters assume that spam emails are fundamentally different than legitimate emails, and can thus be identified by the application of certain heuristics. These heuristics can be divided into three major types: origin-based, traffic analysis-based, and content-based filtering.

Origin based filtering is done by filtering emails before they fully arrive at the computer of the user, which can help reduce the costs commonly attributed to spam emails. There are a few ways of doing this. The first is to use blacklists, which are lists containing addresses of known spam producers, to automatically refuse connections with addresses on the blacklist. This is somewhat effective at blocking known spammers, but can be circumvented by sending mail to servers not on the blacklist, which can then pass the messages on. Another approach is to use whitelists instead. These whitelists contain addresses from which connections are automatically accepted, while the server refuses all other mail. The biggest disadvantage with this, however, is the large false positive rate - many legitimate messages are blocked by the filter.

Traffic analysis-based filtering is done by using the log files of the mail server to detect spam senders by analyzing a number of qualities. These include differences in connection time, amount of mail coming from a host, the number of recipients sent to from a host, and whether or not the message is relayed.

Finally, content-based filtering happens after the message has been received by the user. Once this happens, the content of the message is analyzed to determine whether or not the email is spam. This filtering can be done based on the words in the header, subject, and body of the message, checking known keywords and common features of spam. Multiple algorithms can be used to detect spam in this way, including Naive Bayesian filters, genetic algorithms [10], and neural networks [2]. Our work focuses on Naive Bayes filters, which can be implemented with a number of different modifications.

Naive Bayesian filters work by calculating the probability that a given email is spam based on the probability of the words in the message occurring in spam messages individually. This is done by training the filter on a test set of spam and legitimate (ham) email to determine the probability of any word in the training set occurring in a spam or ham email. Paul Graham [6] does this using the following method:

1. Create a hash map to contain every token in the ham training set, using words as tokens. This hash map will map each token to the number of times they occur in the ham emails.

2. Do the same for every token in the spam training set.

3. Using these hash maps, map each token to the probability that an email containing it is a spam email in a new hash map. This value is calculated using the following equation, where $S(w)$ = the number of occurrences of token w in the spam set, $H(w)$ = the number of occurrences of token w in the ham set, and N_S and N_H are the size of the spam and ham sets, respectively.

$$P(W = w|C = S) = \frac{\frac{S(w)}{N_S}}{\frac{S(w)}{N_S} + \frac{H(w)}{N_H}}$$

Once the probabilities for each word are calculated, the filter can then be used to calculate the probability of a given email being spam, based on the probabilities of each word previously calculated. In general, an email can be treated as a vector $\vec{x} = (x_1, x_2, ..., x_n)$, where $(x_1, x_2, ..., x_n)$ represent the words present in the email. We can categorize this email into category $c \in \{spam, ham\}$ based on this vector using Bayes' Theorem, which gives us the following function [1] [4] [12] [16] [17].

$$P(C = c|\vec{X} = \vec{x}) = \frac{P(\vec{X} = \vec{x}|C = c) * P(C = c)}{P(\vec{X} = \vec{x})}$$

However, calculating $P(\vec{X} = \vec{x}|C = c)$ is difficult and impractical. [1] [4] By making the Naive Bayesian assumption that each feature x_i is independent of every other feature, we find the following equation, which is far easier to calculate.

$$P(\vec{X} = \vec{x}|C = c) = \prod_{i=1}^{n} P(X_i = x_i|C = c)$$

By replacing the original quantity with this new quantity, we gain a final equation for a Naive Bayes approximation of the probability of the email being spam.

$$P(C = c|\vec{X} = \vec{x}) = \frac{\prod_{i=1}^{n} P(X_i = x_i|C = c) * P(C = c)}{P(\vec{X} = \vec{x})}$$

However, doing this for every word in the test email can be both overly complex and misleading. Therefore, we should only pay attention to the most "interesting" tokens. Tokens become interesting if they reach a certain frequency and have probabilities near 1 or 0 of being in a spam email. The filter does this by trimming $\vec{x}$ into the n most interesting tokens and call this set $\vec{m} = (m_1, ..., m_n)$ and using this in the following equation, where S=spam and H=Ham.

$$P(C = S|\vec{X} = \vec{m}) = \frac{\prod_{i=1}^{n} P(X_i = m_i|C = S) * P(C = S)}{\sum_{k \in \{S,H\}} P(C = k) * \prod_{i=1}^{n} P(X_i = m_i|C = k)}$$

Now that we have a probability for the given email of being spam, we can deal with it in two different ways. Because mislabeling a legitimate email as spam (false positive) is much more undesirable than classifying a spam email as legitimate (false negative) [1] [4] [6] [7] [8] [9] [12] [13] [17], we must be err on the side of caution. A message is marked as spam if this probability exceeds a certain threshold T [4] [12] [13]. This threshold can either be a constant or calculated individually for each email. Androutsopolous et al. [1] calculate this threshold by finding the ratio of the probabilities of the message being spam and ham and define λ as the relative cost of false positives to false negatives. Then, if this ratio exceeds *lambda*, we classify it as spam. However, since we know that according to the Theorem of Total Probability, $P(C = H|\vec{X} = \vec{m}) = 1 - P(C = S|\vec{X} = \vec{m})$. Therefore, the threshold T can be found while only calculating one of these probabilities. This allows us to classify an email as spam if the following equation is true.

$$P(C = S|\vec{X} = \vec{m}) > T, where \quad T = \frac{\lambda}{1+\lambda}$$

This threshold can be set anywhere from $\lambda = 1$ to $\lambda = 999$ [1] [16], where higher values of lambda are much less likely to provide false positives.

Now that the basics of Naive Bayesian filters have been explored, we can look at the variations done in previous work.

Metsis et al.[12] look at and compare five different types of Naive Bayes filters. The first type of Naive Bayes, known as Multinomial Naive Bayes is the type discussed above. However, they split this type into two different types: one which counts frequency, Term Frequency; and one which simply checks the presence or absence of tokens, Boolean.

The next type of filter is the Multi-variate Bernoulli Naive Bayes filter. This filter works by treating each message m as a set of tokens, similar to the above approach. However, the main difference is that the probability $P(\vec{x}|C = c)$is calculated using a Bernoulli distribution, rather than a Multinomial one. This function can be seen below.

$$P(\vec{x} \mid C = c) = \prod_{i=1}^{n} P(t_i \mid C = c)^{x_i} * (1 - P(t_i \mid C = c))^{1-x_i}$$

Substituting this into the Naive Bayes equation gives us the following equation.

$$\frac{P(C = S) * \prod_{i=1}^{n} P(t_i \mid C = S)^{x_i} * (1 - P(t_i \mid C = S))^{1-x_i}}{\sum_{k\in\{S,H\}} P(C = k) * \prod_{i=1}^{n} P(t_i \mid C = k)^{x_i} * (1 - P(t_i \mid C = k))^{1-x_i}} > T$$

The third type of Naive Bayes is Multi-variate Gauss Naive Bayes. By assuming that each token of the message follows the normal distribution $g(x_i, \mu_{i,c}, \sigma_{i,c})$ for each category c, where

$$g(x_i, \mu_{i,c}, \sigma_{i,c}) = \frac{1}{\sigma_{i,c}\sqrt{2\pi}} e^{-\frac{(x_i-\mu_{i,c})^2}{2\sigma_{i,c}^2}}$$

and the mean $\mu_{i,c}$ and standard deviation $\sigma_{i,c}$ are estimated from the training data. If we again assume that the values of the tokens are independent, we find the following equation for $P(\vec{x}|C = c)$.

$$P(\vec{x}|C = c) = \prod_{i=1}^{n} g(x_i; \mu_{i,c}, \sigma_{i,C=s})$$

This gives us the final equation for determining if a message is spam as below.

$$\frac{P(C = s) * \prod_{i=1}^{n} g(x_i; \mu_{i,c}, \sigma_{i,C=s})}{\sum_{k\in\{S,H\}} P(C = k) * \prod_{i=1}^{n} g(x_i; \mu_{i,c}, \sigma_{i,C=s})} > T$$

The final type of Naive Bayes is Flexible Bayes. This type allows us to model $P(X_i = x_i|C = c)$ as the average of $L_{i,c}$ normal distributions with different means but the same standard deviation, which gives us the following equation.

$$p(x_i \mid c) = \frac{1}{L_{i,c}} * \sum_{i=1}^{L_{i,c}} g(x_i, \mu_{i,c}, \sigma_{i,c})$$

where $L_{i,c}$ is the number of different values of X_i in the training data for each, which is used as the mean of a normal distribution of that category. Each of these distributions is assumed to have the same standard

deviation $\sigma_c = \frac{1}{\sqrt{M_c}}$, where M_c is the number of training messages in c. A more in-depth description can be found in [12].

When dealing with word frequency, Metsis et al.[12] finds that it can be an effective addition to the filtering process, but makes no distinction in the type of email that is being received. For example, a more formal email may have different frequencies of a specific word over an informal one. A Boolean classification of spam tokens in an email can help to avoid this problem. The article notes that the Boolean Naive Bayes approach performs slightly better than the Term Frequency approach, which tracks the frequency of the token appearing. This seems counter-intuitive, as one would think that providing more information about spam keywords would increase the chances of determining whether or not an email is spam.

O'Brien et al.[13] discuss Naive Bayesian probability method and the Chi by degrees of Freedom method. An interesting variation of the Naive Bayesian method uses characters as tokens instead of using single or multiple words like our approach. Using characters as tokens showed to be very effective, for when characters were used in the Bayesian algorithm the error rate dropped to 0 percent, compared to 13 percent when words as tokens[13]. However, the dataset used for testing by O'Brien consisted of fewer than 500 emails for training and fewer than 50 for testing, where a larger test set would be more conclusive.

The other method of classifying emails discussed by O'Brien et al. is the Chi by degrees of Freedom method. The vast majority of the tens of millions of spam emails are created by as many as only 150 authors [13], so this method seeks to find similarities in spam emails written by the same author. This is calculated by using n-grams which are either characters or words of length n. New emails are compared based on the number of n-gram frequencies that correspond to the new email. The chi by degrees of freedom method when using characters had an error rate of 0.015 when using characters as tokens[13]. This test had only one email that was marked as spam even though it was a legitimate email (a false positive). Chi by degrees of freedom method seemed to perform better than Naive Bayesian method when using words but the opposite trend was shown when the tokens being used where characters for each method. This research shows that the chi by degrees of freedom is an effective approach in dealing with spam filtering as is the Naive Bayesian probability method that our work focuses on.

As stated above, Naive Bayes filtering examines the probability of certain tokens being used in spam emails in order to classify an email as being spam. However, the tokens do not have to be individual words, as seen by multiple authors who utilize phrases of length n, called n-grams, as tokens[5] [8] [9] [17]. When using individual words as tokens, one must assume that the words are independent from each other. This is a naive approach as combinations of words may be more useful in categorizing emails as spam. For example, the phrase "buy now" would probably be more useful in determining that a given email is spam as opposed to using the individual words "buy" and "now" as separate tokens. Most of the experiments conducted are done using n-grams of length 2 and 3, whereas Kanaris et al. uses longer n-grams of length 4 and 5[9]. When using binary methods of Naive Bayes, 4-grams appeared to be the most effective n-gram length. For Term Frequency methods of Naive Bayes probability, there did not appear to be a best n-gram length. This may explain why n-gram lengths of 2 and 3 are more typically used rather than lengths of 4 and 5, since 4 and 5 n-grams suffer from a higher space complexity while only providing marginally improved performance.

Hovold [8] in his work uses Naive Bayes filtering to classify spam emails similar to the others above, but also considers the position of the word in addition to the likelihood that the word belongs to a spam message. His algorithm combines these two characteristics for each word to calculate the probability that an email should be classified as spam as opposed to a legitimate email. It should be noted that the probability for a given word to be at a given location is assumed to be uniform. This means that even though most words may have certain patterns for where they appear in a text, Hovold assumes that they have an equal chance of being at any location. The number of word positions in a document and the number of words kept in the word vocabulary are used to calculate the probability a word has of appearing in a certain type of email. In this case, the two types of email are spam and ham emails. Hovold also explores using n-grams like the others above, but Hovold similarly finds only a minor improvement in performance even with word position.

Our work modifies the previous work done by Metsis et al. [12] by adding the capability of using n-grams as tokens for each of the different types of Naive Bayesian approaches. In doing so, we will be able to determine which type of Naive Bayes works best for the n-grams approach.

3 Approach

To create the training set for the spam filters used in this experiment, we found the frequency of each word in the ham and spam datasets, which was used to estimate $P(w|S)$ for each word. However, words were only included if they exceeded a certain frequency in order to prevent statistical outliers from clouding the filter's judgment. At this step, we also calculated $P(c)$ for both spam and ham. Once these probabilities was calculated, we could use a Niave Bayes method to estimate $P(C = S|X_i = x_i$, or the probability that c is spam given that it contains the vector of words X_i.

We decided to implement three different Naive Bayesian classifiers. The first type we implemented was Multinomial Naive Bayes with Term Frequency Attributes, which utilizes how many times a word is used in the test email to calculate the probability of the test email being spam using the Multinomial Naive Bayes event model. The second type we implemented was Multinomial Naive Bayes with Boolean Attributes. This version also utilizes the Multinomial Naive Bayes event model to calculate the probability, but does not account for the number of times a word is present. The third and final Naive Bayes classifier we implemented was the Multi-variate Naive Bayes classifier. This classifier, like Multinomial Naive Bayes with Term Frequency Attributes, also keeps track of how many times a word is used in the test email to calculate the probability of the test email being spam. However, it uses a different method of calculating the probability itself - Bernoulli Naive Bayes, as opposed to Multinomial.

In addition to having three different types of Naive Bayes filters, we also calculated the probabilities using different lengths of word n-grams. An n-grams is a contiguous sequence of n words that can be used instead of singular words when defining tokens. In our approach, we tried using singular words, 2-grams, and 3-grams as tokens for all three of our different Naive Bayesian classifiers.

For our filter, we chose to use Python as our programming language. We created a few python files in order to both parse through the email data sets and to calculate the ham and spam probabilities of new emails. Python was used because it is a straightforward programming language for reading the email text files.

To test our filter, we used the dataset provided by Metsis et al. [11]. This dataset contains 17171 spam and 16545 ham emails collected from several email users who worked at Enron after the Enron investigation. More information on the origin of these emails can be found here [12]. These emails are stored as text files and are separated based on email type. They are also "preprocessed," meaning that they only contain the subject line and the body of the emails. 1500 of the ham mails and 4500 of the spam mails were used to test the filters, with the rest used as the training set.

This dataset has a collection of separated ham and spam emails in the form of text files. We created a python program that would parse through all of the directories and files to find individual words and/or word sequences depending on the the size of the n-grams. In short, each text file email is read line by line and in each line, the line is separated by white space to create an array of word strings. Then each string was stripped of undesired characters and the count for that word was increased by one. If the word did not already exist in the dictionary, it was added to it instead. A similar tactic is used for n-grams where words were combined together to form n length phrases which were also added to the dictionary of the respective email type. Two dictionaries were created: one ham dictionary and one spam dictionary. Using these two dictionaries, we created ham and spam output files containing one word and its frequency per line in order to pass the data to the next python file. This next file then processed these output text files when determining the likelihood of a given email being ham or spam. We separated these processes so that the dictionaries would not have to be re-created every time the filter was run.

To test an email, the dictionaries were first loaded in from the previously created files and used to calculate $P(c)$ for both spam and ham and $P(W = w|C = S$ for each word, as described above. This was done using the function outlined in the Related Work section. The code for this function appears in the appendix, section 8.2. However, for words that only appeared in the ham or spam set, a value of .01 and .99 were used, respectively. This kept values of 1 and 0 from overpowering the rest of the words during calculation.

Once these probabilities were calculated, each type of Naive Bayes classifier was run on the test set. This entailed finding the most interesting tokens in the test message and using the Naive Bayes classifiers

outlined above. These tokens could be either single words or n-grams. To find these tokens, the test message was parsed using the same procedure used for the training set. To find which tokens were most interesting, $P(W = w|C = S$ for each token in the test email was retrieved, and these tokens were sorted from largest to smallest by the absolute value of their respective probability - .5. This would allow us to find the tokens with probabilities closest to 1 and 0. Of these tokens, the top 20% were used in our calculation.

Each Naive Bayes classifier used a different formula to estimate $P(C = S|X_i = x_i$. These formulas can be found in the Related Work section, while the code used can be found in the appendix, sections 8.3 through 8.5. Once this probability was found, the algorithm marked the message as spam if the probability exceeded .95 ($T = .95$). This was loose enough to get much of the less suspicious spam messages, but still strict enough not to mark many ham messages as spam.

4 Experiment Design and Results

To train the filter, we used a training set containing 12671 spam emails and 15045 ham emails from the Enron dataset [11]. The exact emails used were the spam and ham collections within enron1 to enron5. These files were parsed and tokens were found, treating tokens as n-grams of length 1, 2, and 3 as required. These tokens were added into a dictionary along with their frequency of occurrence among all the messages for each category of message. The full code for parsing the training set can be found in the appendix, section 8.1.

With the frequency of each token in each category of message in hand, we could then test the filter. To do this, we ran each type of Naive Bayes algorithm while varying the n-gram length between 1, 2, and 3 on a set of emails containing 4500 spam emails and 1500 ham emails (enron6). The false positive and false negative rates of each type of filter were found and can be shown in table 1.

Trial Number	Algorithm	ngrams	False Positives	False Negatives
1	Multinomial Naive	1	53 (3.53%)	132 (2.93%)
2	Bayes, Term	2	38 (2.53%)	70 (1.56%)
3	Frequency Attributes	3	42 (2.80%)	76 (1.69%)
4	Multinomial Naive	1	45 (3.00%)	140 (3.11%)
5	Bayes, Boolean	2	28 (1.87%)	72 (1.60%)
6	Attributes	3	34 (2.27%)	81 (1.80%)
7	Multi-variate	1	123 (8.20%)	274 (6.09%)
8	Bernoulli	2	46 (3.06%)	119 (2.64%)
9	Naive Bayes	3	44 (2.93%)	126 (2.80%)

Table 1: Performance of the three Naive Bayesian spam filters with varying n-gram lengths

In total, we ran nine tests, comparing false negative and false positive rates between filters. A result was considered a false positive when a ham message was misclassified as a spam message, and a false negative occurred when a spam messages was misclassified as ham. Because false positives are much worse than false negatives, performance is mostly measured by false positive rate. Multinomial Naive Bayes with Boolean Attributes using n-gram lengths of two (Trial 5) had the lowest false positive rate of 1.87%, while Multi-variate Bernoulli Naive Bayes using n-gram lengths of one (Trial 7) had the highest rate of 8.2%. In terms of false negative rates, Multinomial Naive Bayes with Term Frequency Attributes using n-gram lengths of 2 (Trial 2) had the lowest rate of 1.56%, followed closely by Multinomial Naive Bayes with Boolean Attributes using n-gram lengths of two (Trial 5). The highest rate of false negatives was found when using Multi-variate Bernoulli Naive Bayes using n-gram lengths of one (Trial 7), at a rate of 6.09%.

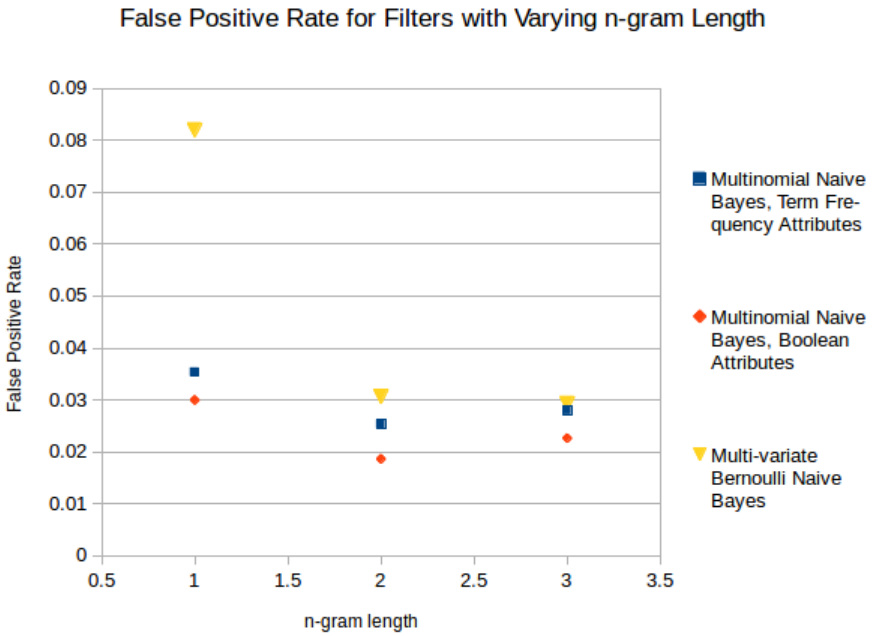

Figure 1: False Positive Rate for Filters

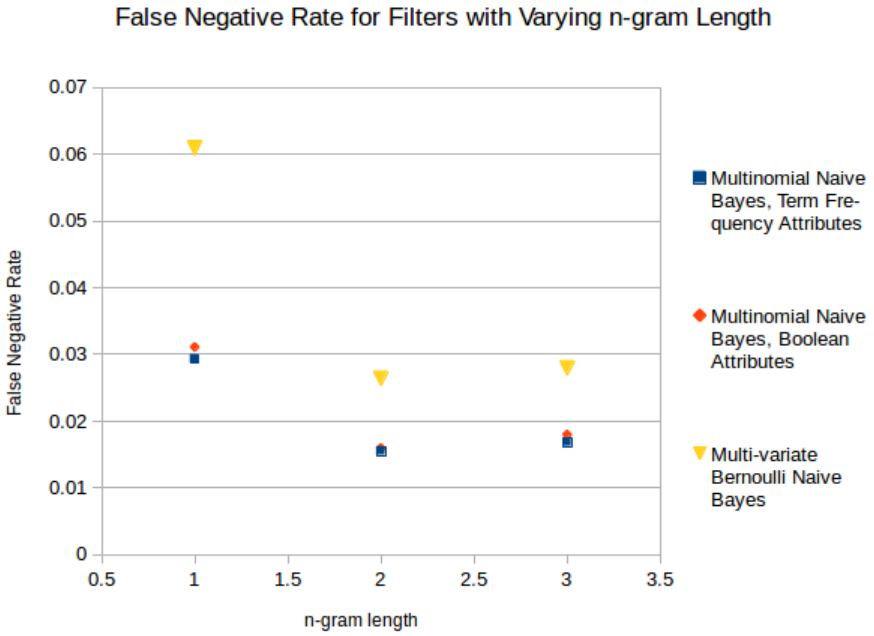

Figure 2: False Negative Rate for Filters

5 Analysis

Table 1 shows the results of each of the three types of Naive Bayes classifiers using different lengths of n-grams. All of the trials were run using the same training set and test set. A value of .95 was used for T,

the threshold at which a message would be marked as spam. Based on our results, Multinomial Naive Bayes with Boolean Attributes using an n-gram length of 2 (Trial 5) had the best performance among the 9 filters. It had the lowest false positive rate of 1.87% (28 of 1500 emails), and the second lowest false negative rate of 1.60% (72 of 4500 emails). The filter with the worst performance was Multi-variate Bernoulli Naive Bayes using n-gram lengths of one (Trial 7), which had the highest rate for both false positives and false negatives.

For the filters that used the Multinomial Naive Bayes classifier, using Boolean attributes rather than Term Frequency improved the performance of the filter. This fact was also noted by Metsis et al.[12], who point out that Boolean attributes may give better performance if the attributes do not follow Poisson distributions. Both of these filters also performed better than Multi-variate Bernoulli Naive Bayes, suggesting that the Multinomial Naive Bayes performs better in general when categorizing messages in this way.

In terms of general trends, using n-gram lengths of two (2-grams) seemed to give the best results for Multinomial Naive Bayes. It is intuitive that the addition of 2-grams would give the filter a performance increase, since the filter would have twice as much information to use when determining the most interesting tokens. However, it can be seen that for Multinomial Naive Bayes, the addition of 3-grams when finding these tokens reduces the performance of the filter. This seems counter-intuitive, given that doing so adds the same amount of information as adding 2-grams. This may be because the same three-word phrases might be repeated more often in ham emails, while spam emails contain more variation on what three word phrases they contain. This would make the frequency of 3-gram tokens in the spam set too low to be meaningful to the filter. For Multi-variate Bernoulli Naive Bayes, however, the addition of 3-grams improves the performance of the filter with regards to false positives. This would imply that the cause lies in the algorithm rather than trends in the datasets. More work is needed to determine the exact cause of this behavior. But, it must be noted that even though adding 3-grams reduces the performance of the filter when compared to 2-grams, it is still a net performance increase when compared to single-word tokens for all three types of Naive Bayes filters. This shows that the addition of n-grams gives better performance and should be implemented, despite the additional space and processing required.

Compared to the results of Paul Graham [6], our results fall somewhat short. In his article, he reports a false positive rate of 0% and a false negative rate of .5%. However, these results are lacking in both size and scope; Graham only runs his filter on his own mailbox and does not release the number of test and training messages used. He does mention that a lot of his emails have programming examples in it which could help to explain his improved performance. In our dataset, we use emails that are preprocessed meaning that we only use text that is found in the email body or the subject line of the email. Graham utilizes the javascript and html tags in an email when he creates his word and n-gram tokens. This could be another explanation for his improved performance in comparision to our test results. Compared to other research groups, however, such as Metsis et al.[12] and Hovold [8], similar rates of false positives and negatives were found. It seems that more effective filters must include more filtering techniques than exclusively Naive Bayesian filtering, such as whitelists and blacklists.

6 Conclusion and Future Work

In this paper we have described several procedures using different Naive Bayes classifiers for filtering spam emails from legitimate emails. We also explored using n-grams with these different types of Naive Bayes spam filters. Our goal was to ultimately to find the spam filter that classifies ham and spam emails with the fewest false negatives and with the fewest false positives. Our results indicate that the addition of n-grams of length 2 and 3 improve performance noticably, with an n-gram length of 2 being the most effective. Of the Naive Bayes classifiers we tested, Multinomial Naive Bayes with Boolean Attributes was found to be the most effective. These results agree with those of Metsis et al.[12].

It might be beneficial to experiment with n-gram lengths greater than 3 in the future for better results. Word sequences longer than 3 words could provide useful phrases that would easily distinguish an email as either ham or spam. Another technique to improve our Naive Bayes filters would be to introduce whitelists and blacklists. These lists could reduce our false positive rates by always classifying emails from reliable email users as ham and false negative rates by always classifying emails from untrustworthy email users as

spam. By classifying emails without doing probabilistic calculations, we could avoid potential failures from our filter. Other Naive Bayes classifiers, such as Multi-variate Gauss Naive Bayes and Flexible Bayes, could also be tested to see if there are other classifiers with a higher performance. In addition to testing other Naive Bayes classifiers, other probabilistic classifiers of the non-Naive Bayesian variety could be tested to classify ham and spam emails. Probabilistic classifiers that are not Naive (meaning that the data such as the words of an email are not independent) would be interesting to test considering that we assume that each word or n-gram is independent. Removing this assumption of words or n-grams being independent from the other surrounding information could improve our performance. We also assume that word position is irrelevant for each word. Implementing the word position techniques employed by Hovold could also reduce the number of false positives and negatives.[8]. These are just a few of techniques and methods that could be added to our selection of spam filters to find a better performing spam filter.

7 Contributions by Group Member

Our work was divided as follows:

- Wrote multiple sections of the report which include: Appendix, Approach, Conclusion, Experiment Design and Results, and Related Work
- Created code to filter spam and ham tokens from an input file
- Wrote code to calculate probabilities using Multinomial Naive Bayes with Term Frequency and with Boolean attributes
- Ran the tests
- Revised document for turn in

- Wrote multiple sections which include: Abstract, Appendix, Approach, Conclusion, Introduction, and Related Work
- Created code to parse through the emails and create the readable data storage for email tokens
- Wrote code to calculate Multi-variate Bernoulli Naive Bayes probabilities
- Revised document for turn in

References

[1] Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., Paliouras, G., and Spyropoulos, C. D. An evaluation of naive bayesian anti-spam filtering. *arXiv preprint cs/0006013* (2000).

[2] Clark, J., Koprinska, I., and Poon, J. A neural network based approach to automated e-mail classification. In *null* (2003), IEEE, p. 702.

[3] Dwork, C., and Naor, M. Pricing via processing or combatting junk mail. In *Advances in Cryptology—CRYPTO'92* (1992), Springer, pp. 139–147.

[4] Garcia, F. D., Hoepman, J.-H., and Van Nieuwenhuizen, J. Spam filter analysis. In *Security and Protection in Information Processing Systems*. Springer, 2004, pp. 395–410.

[5] Graham, P. Better bayesian filtering, 2002.

[6] Graham, P. A plan for spam, 2002.

[7] Hird, S. Technical solutions for controlling spam. *proceedings of AUUG2002* (2002).

[8] Hovold, J. Naive bayes spam filtering using word-position-based attributes. In *CEAS* (2005), pp. 41–48.

[9] Kanaris, I., Kanaris, K., Houvardas, I., and Stamatatos, E. Words versus character n-grams for anti-spam filtering. *International Journal on Artificial Intelligence Tools 16*, 06 (2007), 1047–1067.

[10] Katirai, H. Filtering junk e-mail: a performance comparison between genetic programming and naïve bayes. *Unpublished manuscript: citeseer. nj. nec. com/katirai99filtering. html 10* (1999).

[11] Metsis, V., Androutsopoulos, I., and Paliouras, G. The enron-spam datasets.

[12] Metsis, V., Androutsopoulos, I., and Paliouras, G. Spam filtering with naive bayes-which naive bayes? In *CEAS* (2006), pp. 27–28.

[13] O'Brien, C., and Vogel, C. Spam filters: bayes vs. chi-squared; letters vs. words. In *Proceedings of the 1st international symposium on Information and communication technologies* (2003), Trinity College Dublin, pp. 291–296.

[14] Prakash, V. V. Vipul's razor. *URL: http://razor. sourceforge. net* (2007).

[15] Rao, J. M., and Reiley, D. H. The economics of spam. *The Journal of Economic Perspectives 26*, 3 (2012), 87–110.

[16] Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop* (1998), vol. 62, pp. 98–105.

[17] Schneider, K.-M. A comparison of event models for naive bayes anti-spam e-mail filtering. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1* (2003), Association for Computational Linguistics, pp. 307–314.

[18] Tatyana Shcherbakova, Maria Vergelis, N. D. Spam and phishing in q3 2015, 2015. Accessed: 2016-04-23.

[19] Vixie, P., and Rhyolite, L. Distributed checksum clearinghouse, 2007.

8 Appendix

8.1 Training set Parse Function

```
#define dictionary as a new dictionary which will contain all tokens
#define ntokens as an integer from 1 to 3 representing n-gram length

for each training email:
  if (ntokens >= 1):                #for n-grams of length 1
    for line in fileObject:
      tokens = line.split()
      for token in tokens:
        token = token.rstrip(')?:.,-*;"\'\n')   #remove punctuation
        token = token.lstrip('(.@,-?;:*"\'\n')  #characters and
        token = token.replace('\0', "")         #invalid characters
        if token == "":
          continue
        elif token in dictionary:
          dictionary[token] += 1
        else:
```

```
        dictionary[token] = 1

  if(ntokens >= 2):                #for n-grams of length 2
    fileObject.seek(0)
    prevToken = ""                 #variable to hold last valid word
    for line in fileObject:
      tokens = line.split()
      for token in tokens:
        token = token.rstrip(')?:.,-*;"\'\n')   #remove punctuation
        token = token.lstrip('(.@,-?;:*"\'\n')  #characters and
        token = token.replace('\0', "")         #invalid characters
        if token == "":
          continue
        elif prevToken == "":
          prevToken = token
        elif (prevToken + " " + token) in dictionary:
          twoToken = prevToken + " " + token
          dictionary[twoToken] += 1
          prevToken = token
        else:
          twoToken = prevToken + " " + token
          dictionary[twoToken] = 1
          prevToken = token

  if(ntokens >= 3):                #for n-grams of length 3
    fileObject.seek(0)
    prevToken1 = ""                #variables to hold previous words
    prevToken2 = ""
    for line in fileObject:
      tokens = line.split()
      for token in tokens:
        token = token.rstrip(')?:.,-*;"\'\n')   #remove punctuation
        token = token.lstrip('(.@,-?;:*"\'\n')  #characters and
        token = token.replace('\0', "")         #invalid characters
        if token == "":
          continue
        elif prevToken2 == "":
          prevToken2 = token
        elif prevToken1 == "":
          prevToken1 = prevToken2
          prevToken2 = token
        elif (prevToken1 + " " + prevToken2 + " " + token) in dictionary:
          if prevToken1 == "":
            prevToken1 = prevToken2
          elif prevToken2 == "":
            prevToken2 = token
          else:
            threeToken = prevToken1 + " " + prevToken2 + " " + token
            dictionary[threeToken] += 1
            prevToken1 = prevToken2
            prevToken2 = token
        else:
          if prevToken1 == "":
            prevToken1 = prevToken2
          elif prevToken2 == "":
            prevToken2 = token
          else:
            threeToken = prevToken1 + " " + prevToken2 + " " + token
            dictionary[threeToken] = 1
            prevToken1 = prevToken2
            prevToken2 = token
```

8.2 Finding $P(W = w|C = S$ for Each Token in Training Set

```
#define wordsSpam as the dictionary containing P(W=w|C=S) for each word
```

```
#spam is the dictionary conatining token : frequency pairs for the spam set
#ham is the dictionary conatining token : frequency pairs for the ham set

spamLength = len(spam)
hamLength = len(ham)

sumSpam = 0
for key in spam:
  sumSpam += spam[key]

sumHam = 0
for key in ham:
  sumHam += ham[key]

for key in spam:
  n = 0
  inHam = False
    if key in ham:
      n = spam[key] + ham[key]
      inHam = True
    else:
      n = spam[key]
    if n > 5: #only count words that occur more than 5 times total
      if inHam:
        top = float(1+spam[key])/float(spamLength+sumSpam)
        bottom = float(1+spam[key])/float(spamLength+sumSpam)
                 + float(1+ham[key])/float(hamLength+sumHam)
        wordsSpam[key] = top / bottom
      else:
        wordsSpam[key]=.99 #value for words only in spam

for key in ham:
  if key not in wordsSpam and ham[key] > 5:
    wordsSpam[key] = 0.01 #value for words only in ham
```

8.3 Multinomial Naive Bayes with Term Frequency Attributes code

```
#miw is an array containing the most interesting words of the test email
#duplicates is a dictionary containing word:frequency pairs for each token
pcs = float(numSpam)/float(numSpam+numHam)
pch = float(numHam)/float(numSpam+numHam)
productSpam = 1.0
productHam = 1.0
for word in miw:
  productSpam *= (wordsSpam[word[0]] ** duplicates[word[0]])
  productHam *= ((1-wordsSpam[word[0]]) ** duplicates[word[0]])
p = pcs * productSpam / (pcs * productSpam + pch * productHam)
if p > .95: #T=.95
  print "spam"
else:
  print "ham"
```

8.4 Multinomial Naive Bayes with Boolean Attributes code

```
#miw is an array containing the most interesting words of the test email
pcs = float(numSpam)/float(numSpam+numHam)
pch = float(numHam)/float(numSpam+numHam)
productSpam = 1.0
productHam = 1.0
for word in miw:
  productSpam *= wordsSpam[word[0]]
  productHam *= (1-wordsSpam[word[0]])
p = pcs * productSpam / (pcs * productSpam + pch * productHam)
if p > .95: #T=.95
```

```
  print "spam"
else:
  print "ham"
```

8.5 Multi-variate Bernoulli Naive Bayes code

```
#miw is an array containing the most interesting words of the test email
#duplicates is a dictionary containing word:frequency pairs for each token
pcs = float(numSpam)/float(numSpam+numHam)
pch = float(numHam)/float(numSpam+numHam)
productSpam = 1.0
productHam = 1.0
for word in miw:
  productSpam *= (wordsSpam[word[0]] ** (duplicates[word[0]]\%2)) *
    ((1 - wordsSpam[word[0]]) ** (1 - (duplicates[word[0]]\%2)))
  productHam *= ((1-wordsSpam[word[0]]) ** (duplicates[word[0]]\%2)) *
    (wordsSpam[word[0]] ** (1 - (duplicates[word[0]]\%2)))
p = pcs * productSpam / (pcs * productSpam + pch * productHam)
if p > .95: #T=.95
  print "spam"
else:
  print "ham"
```