**CIS 3700 – ASSIGNMENT 5**

**Irenaeus Chan**

**0797453**

**Bayesian Spam Filtering**

**Dataset of spam e-mail messages and a dataset of non-spam email messages (aka ‘ham’).**

Test cases were pulled from <https://spamassassin.apache.org/publiccorpus/> utilizing their, 2005 and 2003 spam collection and 2003 hard and easy ham collection. Each data set included approximately 1200 emails confirmed to be either spam or ham. From these datasets, the first 600 emails from each dataset were separated into learning and testing cases. The 2400 emails used for learning (both spam and ham) were not randomized.

**Tokenize your datasets into words (decide on suitable delimiters: spaces, new-lines, punctuation…).**

The tokenization of the dataset was decided to be based purely on words ignoring all special characters, punctuation and spaces. In addition, capitalization was also ignored, to account for words such as “The”, “the”, and “THE” being considered distinct. Therefore all words such as “The”, “the”, and “THE” would be considered part of the count “the”. Special characters were ignored because the frequency at which they would normally appear in a given sentence for either spam or ham emails would be infrequent, which is what prompted my choice in ignoring them all together. Punctuation was ignored because I wanted to mainly focus on keywords that would appear in spam emails compared to ham emails. There are special cases for both of these exceptions such as large sums of money “$1,000,000,000” and “????????????” would not be counted properly even though they may be indicative of spam, however I decided there should be more intricate ways of detecting spam aside from these obvious cases.

*spam.py*

This script will create 2 dictionaries utilizing the two provided test cases “*learning\_ham*” and “*learning\_spam*”. After creating the two dictionaries, the script will then create two files titled “*outputHam.txt*” and “*outputSpam.txt*” which contains the total number of words matched for all data sets, a list of all the words, their frequencies, P(word|spam or ham), and P(spam or ham|word). To use this script, ensure that learning datasets are in the current directory, then on the command line type: python spam.py. This will produce the two files stated above.

**Build a dictionary listing all the words that occur in each dataset along with the number of times that each word occurs in each dataset.**

Attached to this file will be two textfiles titled “*outputSpam.txt*” and “*outputHam.txt*”. Each file contains the total number of words matched for all data sets at the start of the file and a list of all words, their frequencies, the probability of that word appearing in the given dataset P(word | spam) or P(word | ham), and the probability of spam or ham given that word.

**Using these frequencies as estimates of the probabilities that a spam/ham message contains the word, calculate the P(word | spam) and P(word | ham).**

The probability of the word appearing in the dataset was calculated by taking the frequency of the words appearance in the learning dataset and dividing it by the total number of words matched all together. This probability was then listed beside the frequencies in the attached files “*spam\_results.txt*” and “*ham\_results.txt*”.

**Estimate which percentage of messages sent to your email are spam/ham.**

As I utilize gmail for most of my active emails it is difficult to estimate a given number due to the effectiveness of Google’s spam protection. After doing a bit of research, it was discovered that in the earlier years of email messaging the percentage of spam emails sent were an astounding 80% only recently falling below 70% in 2013. Thus, I estimate that the P(spam) is around 75% and the P(ham) is around 25% due to my test data being significantly older than the current established numbers, and the lack of statistics available for these years.

*Sources:*

<http://en.wikipedia.org/wiki/Email_spam>

<http://www.kaspersky.com/about/news/spam/2013/Spam_in_Q2_2013_More_offices_in_danger_from_targeted_plausible_fakes>

**Using Baye’s theorem compute P(spam|word) and P(ham|word) for each word. Ignore cases where a word occurs infrequently in both datasets.**

Cases where the total amount of words for a specific word was less than 10 was considered infrequent and therefore removed by returning the probability to be 50%, assuming it has a likely chance of being in both spam and or ham category. For the rest of the cases Baye’s theorem was calculated normally using the formula:

\Pr(S|W) = \frac{\Pr(W|S) \cdot \Pr(S)}{\Pr(W|S) \cdot \Pr(S) + \Pr(W|H) \cdot \Pr(H)}

Where S = spam, W = word, H = ham. For certain words that only appeared in Spam or only appeared in Ham and appeared more than 10 times, these words were assumed to be distinct specifically to the given category, and thus, 90% probability was given for words only appearing in Spam and 10% probability was given for words only appearing in Ham, due to the chance that the test data wasn’t large enough to encompass all possible words. Again, this probability was then listed beside the word probabilities in the attached files “*spam\_results.txt*” and “*ham\_results.txt*”.

*Source:*

<http://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering#Computing_the_probability_that_a_message_containing_a_given_word_is_spam>

**Develop a method for computing P(spam|word1 and word2 and word3…) and P(ham|word1 and word2 and word3…).**

The likelihood for a message to be spam clearly does not lay in a single word as a defining keyword, but an entire string of words all notably keywords. Thus the combination of all these probabilities must be considered. In order to determine whether or not a message is spam or ham we would have to take into consideration the probability of the first word hinting of spam, as well as the second word, and the third, and so on and so forth. Chaining these probabilities together, we can reach an outcome in which, the final probability will be a total of all the other word probabilities giving us a good estimation of where it belongs. The equation for this combination of probabilities is as follows:

p = \frac{p_1 p_2 \cdots p_N}{p_1 p_2 \cdots p_N + (1 - p_1)(1 - p_2) \cdots (1 - p_N)}

Where p1 = probability of Word 1 appearing in spam, p2 = probability of Word 2 appearing in spam, etc. However, this organization of this equation can lead to some problems mostly concerning the floating-point underflow in which there is a possibility for a rare occurrence of a word creating a 0 value which would undermine the entire probability due to the nature of multiplicity. Thus a different algorithm:

\eta = \sum_{i=1}^N \left[ \ln(1-p_i) -\ln p_i \right]   p = \frac{1}{1 + e^\eta} 

was used to calculate the combined probability of all the words in a given message being either spam or ham. Although there are some cases in which the probability of a word would still equal 0 based on the calculation of P(Spam|Word), this was accounted for as stated earlier in words that were infrequent as well as rare words and words that only appeared in a single dataset.

*Source:*

<http://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering#Computing_the_probability_that_a_message_containing_a_given_word_is_spam>

<https://shapedbychance.wordpress.com/2013/07/22/spam-vs-ham/>

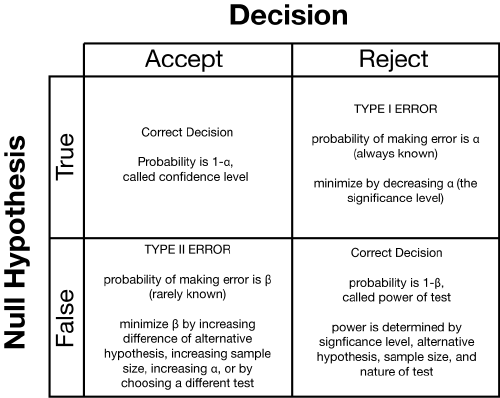
**Build a Bayesian spam filter based on the above that classifies email messages as either spam or ham.**

Using the combined probability as stated above, it is possible to compare this probability of whether or not a group of words in a message is spam or ham to a defined threshold or what statisticians like to call the null hypothesis test. In which, if the probability is greater than a defined significance level, then we can accept the test or the probability to be true, even if there is a margin for error. Depending on this threshold, spam emails could either all be stopped, but risk the possibility of blocking ham emails, or some spam emails could potentially get through, but all ham emails are allowed through.

*spam.py*

In addition to learning the two dataset provided, the spam.py program also comes with the function to determine whether or not a folder containing email messages is considered spam or not. Simply going into the program and choosing the correct path for the variable fileTest will allow users to receive a list of files within the test folder that are either SPAM or HAM depending on the set confidence level.

**Adjust your filter considering how you feel about the importance of detecting spam vs finding false positives.**



For this Bayesian spam filter, the threshold that I decided upon was 90%. If the combined probability of all the words in the message passed the threshold of 90%, then I would accept it as spam. Of course, this is lower than normal confidence levels which utilize a 95% threshold. However, I personally believe using a 90% threshold level is fairly good. Anything lower than a 90% threshold and a lot more Ham emails may be identified as Spam, especially if they are more spam-like in nature such as a job inquiry. Increasing it however would make the test stricter potentially allowing Spam emails through that are better disguised. Originally the threshold was set at 95%, but it ended up being far too high and many Spam messages was able to get through due to the confidence level being too high rejecting a lot of spam messages that looked a lot like normal ham messages, thus I ended up lowering the threshold to 90% in order to account for less predictable Spam emails. I personally believe that stopping Spam messages is more beneficial than preventing a few Ham messages from being blocked. Personally speaking, if an email or a message is important and it does end up in the Spam folder, said sender will typically inquire about the message which would prompt me to check. This is better than having 10 new emails appear in the inbox, and 9 of them is spam. While there are cases such as the court of law in which false positives are looked down upon, many things such as vaccinations and apparently email filter allow more leniency when it comes down to false positives. But again, this ends up being a user preference, which is why the filter’s confidence level can be changed depending on the needs of the user.

**Evaluate the accuracy of your filter based on the dataset that you used to construct the filter. Evaluate the accuracy of your filter based on a dataset of messages that were not used to construct the filter.**

Table 1: Summary of the 6 test cases used during the construction of the Bayesian Spam Filter

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Number of Messages** | **Correctly Identified** | **Incorrectly Identified** |
| Learning\_Spam | 1396 | 1229 | 167 |
| Learning\_Ham | 1400 | 1365 | 35 |
| Test\_Spam | 1397 | 1229 | 168 |
| Test\_Ham | 2501 | 2327 | 174 |
| Hard\_Ham | 250 | 86 | 120 |
| 5050\_Test | 712 | 640 | 72 |

As seen above, there were over 7600 messages tested for the Bayesian Spam Filter. For the majority of the easier cases specifically the cases used during the learning process, the success rate was fairly high, 88% and 97.5% for *Learning\_Spam* and *Learning\_Ham* (Table 1). When I moved on to unseen test cases there was a slight decrease in success but overall the results were surprisingly high, 87.9% and 93% for *Test\_Spam* and *Test\_Ham* (Table 1). However, when testing against harder cases in which Ham seems very similar to spam-like messages, the rates were much lower than what I had originally predicted being only a 34% success rate overall (Table 1). However, when looking at a *50/50 Test Case* of mixed cases, the results were again around 90% (Table 1). Based on these results, I can safely say the Bayesian Spam Filter does work for the majority of normal messages. While, yes there are some cases in which the filter will block ham messages, the overall success of getting Ham messages correct is high enough to warrant the lower Spam messages. However, based on these results, I personally feel that I should have an even lower confidence level in order to account for more spam messages, but I do believe this is personal preference.

**Conclusion**

In conclusion, I personally believe this Spam filter is fairly good. For most normal everyday messages the Spam Filter can clearly distinguish between Spam and Ham with approximately 90% accuracy. However, when looking at harder Ham messages, the fact that the success percentage dropped to 34% is concerning, which prompts future considerations into the program itself to utilize a better way of breaking down the learning cases possibly consider punctuation and special characters or looking at grammar structure or possibly reconsider the 50% threshold and value Spam emails more than Ham emails. There are countless ways to improve the system, but for the normal everyday use, I personally believe that the effectiveness of this spam filter is great.