Bayesian Spam Filtering

Ahmed Obied

Department of Computer Science
University of Calgary
amaobied@ucalgary.ca
http://www.cpsc.ucalgary.ca/~amaobied

Abstract. *With the enormous amount of spam messages propagating on the Internet these days, anti-spam researchers and developers are trying to build spam filters to get rid of such unsolicited messages as accurately as possible. In this paper, I describe a machine learning approach based on Bayesian analysis to filter spam. The filter learns how spam and non-spam messages look like, and is capable of making a binary classification decision (spam or non-spam) whenever a new email message is presented to it. The evaluation of the filter showed its ability to make decisions with high accuracy (96.24% in the worst case and 99.66% in the best case).*

Key words: Machine Learning, Text Classification, Spam

Table of Contents

List of Figures

1 Introduction

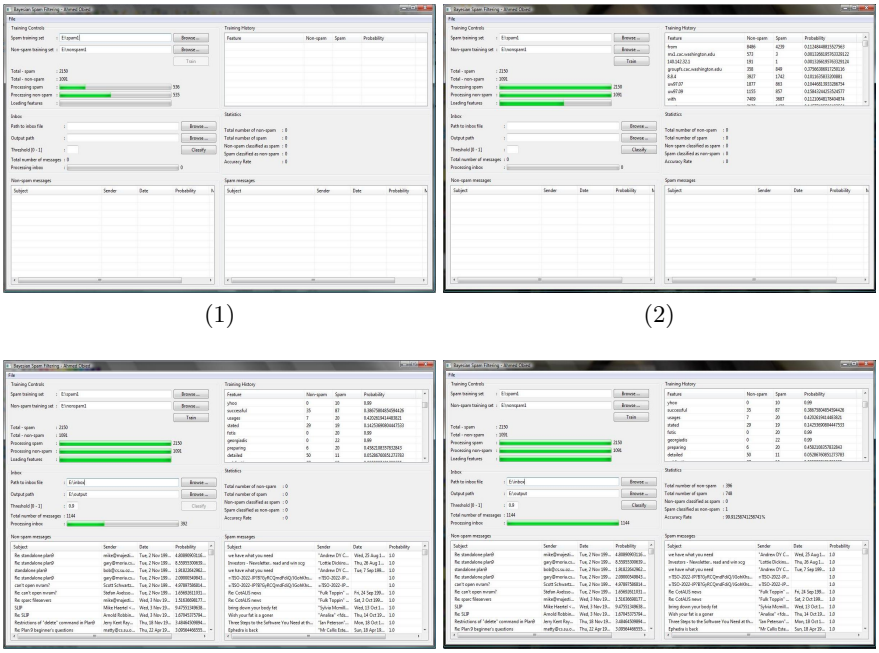
In today's highly technical world and our computer-connected society, email has become the fastest and most economical form of communication available. The widespread use of email enticed direct marketers to bombard unsuspecting email inboxes with unsolicited messages regarding everything from items for sale and get-rich-quick schemes to information about accessing pornographic Web sites [8]. The volume of these unsolicited messages (referred to as spam messages) has grown tremendously in the past few years. Many anti-spam researchers believe that spam is responsible for anywhere from 35% to 65% of all email traffic on the Internet today, with a whopping annual growth rate of 15% to 20% [10]. Many are concerned that spam could be the end of email [10].

A precise definition for spam does not exist since spam is capable of evolving. To many, spam is defined as unwanted email messages. To others, spam is defined as unsolicited commercial email (UCE) or unsolicited bulk email (UBE). In the past few years, a number of anti-spam techniques have been proposed and used to combat spam such as whitelisting, blacklising, and greylisting of domain names, keyword-based filtering, heuristic-based filtering, etc. All of these techniques, however, require heavy maintenance and can not achieve very high overall accuracy. The best heuristic-based spam filter available on the Internet today can not achieve more than a 95% accuracy. Moreover, such accuracy drops substantially when spam evolves. To overcome the disadvantages of the anti-spam techniques described above and achieve very high accuracy, one can use machine learning techniques. Spam filters that are based on machine learning techniques are capable of learning, let the user define what spam is and generate few false positives. Moreover, such filters adapt easily with the way spam evolves and are capable of achieving an accuracy rate higher than 95% [10] if the filter is well trained.

2 Approach

The process of identifying the disposition of a presented text into a particular category is known as text classification [10]. I implemented a statistical text classifier (filter) in Java that is based on Bayesian analysis. The filter classifies an email message into one of two categories: spam or non-spam. I used a supervised learning approach to enable the filter to build a history of what spam and non-spam messages look like. Figures 1 shows screen shots of the filter that was implemented. Give two classes $C =< C_1 = spam, C_2 = non - spam >$ and features $f_1, f_2, ..., f_n$, the probability that these features belong to a certain class using naive Bayesian can be expressed as follows:

$$P(C|f_1, f_2, ..., f_n) = \frac{P(f_1, f_2, ..., f_n|C)P(C)}{P(f_1, f_2, ..., f_n)}$$



(1) (2)

(3) (4)

Fig. 1. Spam Filter

Assuming conditional independence, one can compute $P(f_1, f_2, ..., f_n|C)$ as follows:

$$P(f_1, f_2, ..., f_n|C) = \prod_{i=1}^{n} P(f_i|C)$$

To classify an email message as spam, one can check if it exceeds a specific threshold:

$$\frac{P(C_1 = spam|f_1, f_2, ..., f_n)}{P(C_2 = non - spam|f_1, f_2, ..., f_n)} >= \beta$$

2.1 Dataset

I found a number of public spam corpora that can be used to train and test a spam filter. The corpus I used to train the filter and test it is the TREC 2006 Spam Track Public Corpora [1] since it contains a large number of email messages. The corpus contains 37,822 email messages; 12,910 are non-spam messages and 24,912 are spam messages.

The problem with the TREC corpus is that the non-spam and spam messages are not separated into two different directories. Both types of messages are placed in the same directory. The only way to know if a message is spam or non-spam is by looking at an index file which has the message number and its type. Since having the spam and non-spam messages in two separate directories can make the training and testing easier, I decided to write a program in C to move the spam messages into a spam directory and the non-spam messages into a non-spam directory. Every time a message is moved to a specific directory, the program re-writes the file name of the message to include a letter in it (n for non-spam messages and s for spam messages). These letters in the file names are used later in the testing phase to identify the messages that have been classified correctly and incorrectly. The program also checks if a message is encoded in base-64 and drops it if it is indeed encoded in base-64. Base-64 is used to encode email attachments. Since such encoding does not increase the effectiveness of the filter during the training phase but can decrease it, I decided to drop such messages.

After processing the corpus, I ended up with a total of 36,391 email messages; 12,594 non-spam messages and 23,979 spam messages. I then divided the corpus into two corpora: a training corpus and a testing corpus. The training corpus contains 70% of the original corpus: 25,474 email messages; 8,816 non-spam messages and 16,658 spam messages. The testing corpus contains 30% of the original corpus: 10,917 email messages; 3,778 non-spam messages and 7,139 spam messages.

2.2 Feature Extraction

The feature extractor in the filter is used to extract features $v_j = f_1, f_2, f_3, ..., f_n$ from an email message and construct feature vectors $V =< v_1, v_2, v_3, ..., v_i >$ that can be used in the classification phase. A feature can be anything in an email message. It can be a word, a phrase, a number, an HTML tag, etc. The way the features are represented can affect the filter's accuracy to some extent.

An email message contains two parts: a header and a body. The header consists of fields usually including at least the following:

- From: This field indicates the sender's email address.
- To: This field indicates the receiver's email address.
- Date: This field indicates the date and time of when the message was sent.
- Subject: This field indicates a brief summary of the message's content.
- Received: This field indicates the route the message took to go from the sender to the receiver.
- Message-ID: This field indicates the ID of the message. Every email message has a unique ID at a given domain name: *id@senders-domain-name.*

On the other hand, the body of an email message contains the actual content of the message. The header and the body of an email message is separated by an empty line. To extract features from an email message, I do the following:

1. Parse an email message and extract the header and the body parts.

2. Remove the fields from the header of the message since they appear in every email message.
3. Extract features from the header by tokenizing the header using the following delimiter: \n\f\r\t\ /&%#_{}[]! +=-() ‘”*?:;<>
4. Extract features from the body by tokenizing the body using the following delimiter: \n\f\r\t\ ./&%#_{}[]! +=-() ‘”*?:;<>
5. Ignore features of size strictly less than 3 and digits.

To store the features in the training phase, I use a hash table of nodes. Each node stores the following information:

- Number of occurrences in spam messages (initialized to 0).
- Number of occurrences in non-spam messages (initialized to 0).
- Probability (initialized to 0).

Every time a feature is extracted, I check if the email is spam or non-spam. If it is spam then the number of spam occurrences is incremented by one. Otherwise, if it is non-spam then the number of non-spam occurrences is incremented by one. After extracting all the features from the training set and filling the hash table with such features, I start enumerating through the elements of the hash table to compute the probability for each feature. To compute the probability P for a feature f, I use the following formula:

$$P_f = \frac{\frac{s}{t_s}}{\frac{s}{t_s} + \frac{kn}{t_n}}$$

s is the number of occurrences of feature f in spam messages, t_s is the total number of spam messages in the training set, n is the number of occurrences of feature f in non-spam messages, and t_n is the total number of non-spam messages in the training set. k is a number that can be tuned to reduce false positives by giving a higher weight to number of occurrences of non-spam features. In my implementations, I set k to 2. If a feature f only appears in a single corpus (spam or non-spam) then I assign it a specific value. If it appears only in spam messages then I assign it 0.99. On the other hand, if it appears only in non-spam messages then I assign it 0.01. All the features have probabilities between 0 and 1 exclusively. The closer the probability of a given feature is to 1, the more spammy such feature is and vice versa.

2.3 Classification

The analysis engine is the core of the filter. It is the part that applies Bayes theorem based on Graham's assumption [4] that there is no prior knowledge of spam and non-spam messages, and computes the probability distribution of features in a given email message. After training the filter and having all the necessary information stored in the hash table, the filter becomes capable of making decisions based on what it has seen in the training set. Every time an email message is presented to the filter in the classification phase, I do the following:

1. Parse a given email message and extract the header and the body parts.
2. Extract features from the header and body using the feature extraction method described above.
3. Look up the features in the hash table and retrieve their corresponding probabilities. If a feature is not in the hash table then this means that it is a feature that the filter has not seen in the training set. In this case, I assign it a probability of 0.5. After looking up the features and retrieving their corresponding probabilities, I compute the interestingness of the features. The interestingness I_f for a feature f can be computed as follows:

$$I_f = |0.5 - P_f|$$

4. Extract the most interesting features from the list of features. To do that, I sort the list by interestingness in descending order and take the first N features. The value of N that I used is 15 since it has been shown to be the best choice [10].
5. Combine the probabilities of the N most interesting features using Bayes theorem and based on Graham's assumptions [4]:

$$P = \frac{P_{f_1} P_{f_2} P_{f_3} ... P_{f_N}}{P_{f_1} P_{f_2} P_{f_3} ... P_{f_N} + (1 - P_{f_1})(1 - P_{f_2})(1 - P_{f_3}) ... (1 - P_{f_N})}$$

At the end, I get P which is a value between 0 and 1. The closer the P is to 0, the more likely the message is non-spam and the closer P is to 1, the more Likely the message is spam. The threshold I used to determine whether a given email message is spam is 0.9, that is, if $P >= 0.9$ then I classify the email message as spam otherwise I classify it as non-spam.

2.4 Testing

To make the testing of the classification easier, I included a built-in testing functionality in the filter. The filter takes as input a directory specified by the user that contains email messages (spam and non-spam). The output of the classification is a directory that is also specified by the user. When the filter starts the classification, two directories are created in the output directory: a spam directory and a non-spam directory. If the filter classifies a given email message as spam then the filter copies it from the input directory to the spam directory. Similarly, if the filter classifies a given email message as non-spam then the filter copies it from the input directory to the non-spam directory.

I mentioned earlier in the dataset section that every message has a letter in its file name: s if the email message is spam and n if the email message is non-spam. When the filter completes the classification process, it traverse the spam and non-spam directories in the output directory to count the total number of spam messages s_m that have been classified as non-spam and the total number of non-spam messages n_m that have been classified as spam. To calculate the total number of non-spam messages that have been classified as spam, the filter traverses the spam directory and increments a counter n_m (initialized to 0) by 1

every time it encounters an email message that has the letter n in its file name. To calculate the total number of spam messages that have been classified as non-spam, the filter traverses the non-spam directory and increments a counter s_m (initialized to 0) by 1 every time it encounters an email message that has the letter s in its file name.

After calculating the values of n_m, s_m, and knowing the total number of email messages in the input directory t, I calculate the accuracy A of the filter in percentage using the following formula:

$$A = (\frac{t - (n_m + s_m)}{t}) * 100$$

2.5 Evaluation

To evaluate the filter's performance, I performed the following test cases:

Test Case 1 The inbox contains 5,000 email messages: 2,500 non-spam messages randomly chosen from the training set and 2,500 spam messages randomly chosen from the training set.

Total number of email messages: 5,000
Total number of non-spam messages: 2,500
Total number of spam messages: 2,500
Total number of email messages classified as non-spam: 2,517
Total number of email messages classified as spam: 2,483
Total number of non-spam messages classified as spam messages: 0
Total number of spam messages classified as non-spam messages: 17
Accuracy: 99.66%

Test Case 2 The inbox contains 5,000 messages: 1,250 non-spam messages randomly chosen from the training set, 1,250 non-spam messages randomly chosen from the testing set, 1,250 spam messages randomly chosen from the training set, and 1,250 spam messages randomly chosen from the testing set.

Total number of email messages: 5,000
Total number of non-spam messages: 2,500
Total number of spam messages: 2,500
Total number of email messages classified as non-spam: 2,549
Total number of email messages classified as spam: 2,451
Total number of non-spam messages classified as spam messages: 27
Total number of spam messages classified as non-spam messages: 76
Accuracy: 97.94%

Test Case 3 The inbox contains 5,000 messages: 2,500 non-spam messages randomly chosen from the testing set and 2,500 spam messages randomly chosen from the testing set.

Total number of email messages: 5,000
Total number of non-spam messages: 2,500
Total number of spam messages: 2,500
Total number of email messages classified as non-spam: 2,606
Total number of email messages classified as spam: 2,394
Total number of non-spam messages classified as spam messages: 41
Total number of spam messages classified as non-spam messages: 147
Accuracy: 96.24%

Test Case 4 The inbox contains 5,000 messages: 2,500 non-spam messages randomly chosen from the testing set, 2,500 spam messages randomly chosen from the testing set.

Total number of email messages: 5,000
Total number of non-spam messages: 2,500
Total number of spam messages: 2,500
Total number of email messages classified as non-spam: 2,590
Total number of email messages classified as spam: 2,410
Total number of non-spam messages classified as spam messages: 20
Total number of spam messages classified as non-spam messages: 110
Accuracy: 97.39%

2.6 Challenges

The main challenge that I found while working on the project was dealing with memory. When you train the filter with enormous amount of email messages, storing all the features in memory might not be the best solution. At first, I was running out of heap space. However, I tried increasing the heap size (which can be done in Java at run time using the -Xms and -Xmm flags) and everything worked fine. Although I tried using a database (MySQL) to store the features at the beginning of my implementation, the access time required to store or retrieve a feature was taking a long time which made me end up storing everything in main memory. In the future, if one wants to deal with large training datasets then the best solution would be to store everything on disk and use a technique similar to paging. A hash table of size n can be stored in main memory. If the number of features in the hash table reaches n then the data in the table can be swapped to disk. The data on disk can be loaded to main memory in a FIFO fashion any time the filter tries to classify a given email message. Although loading the data from disk to main memory and then accessing main memory is slower than accessing main memory directly, it is definitely better and faster than storing the data in a database and querying it.

2.7 Conclusion

More than half of all email traffic we see on the Internet these days is spam. Although a number of anti-spam techniques have been used to counter spam, none of these techniques have the potential to deal with the way spam evolves over time except the anti-spam techniques that are based on machine learning approaches. These techniques are essentially text classifiers but instead of classifying a given text into different categories, they classify a given text (email message) into two categories (spam or non-spam).

In this paper, I described a machine learning approach based on Bayesian analysis to filter spam. The filter learns of what spam and non-spam messages look like and can make binary classification decisions (spam or non-spam) based on what it has learned. The filter does not require any heavy maintenance. All what you need to do is train it once and you are done. After training the filter, it becomes capable of filtering spam with high accuracy as shown in the evaluation section.

References

1. TREC 2006 Spam Track Public Corpora, `http://plg.uwaterloo.ca/~gvcormac/treccorpus06/about.html`.
2. W. Cohen. Learning Rules that Classify Email. In *Proc. of the AAAI Spring Simposium on Machine Learning in Information Access*, pages 124–143. 1996.
3. P. Graham. A Plan for Spam. 2002.
4. P. Graham. Better Bayesian Filtering. 2003.
5. P. Graham. Filters that Fight Back. 2003.
6. E. Michelakis, I. Androutsopoulos, G. Paliouras, G. Sakkis, and P. Stamatopoulous. Filtron: A Learning-Based Anti-Spam Filter. In *Proc. of the 1st Conference on Email and Anti-Spam (CEAS 2004), Mountain View, CA, USA*, 2004.
7. C. Pu, S. Webb, O. Kolesnikov, W. Lee, and R. Lipton. Towards the Integration of Diverse Spam Filtering Techniques. In *Proc. of IEEE International Conference on Granular Computing*, pages 7 – 10, 2006.
8. M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *AAAI-98 Workshop on Learning for Text Categorization, Madison, Winsconsin, USA*, pages 55–62, 1998.
9. S. Webb, J. Caverlee, and C. Pu. Introducing the Webb Spam Corpus: Using Email Spam to Identify Web Spam Automatically. In *Proc. of the Third Conference on Email and Anti-Spam (CEAS 2006)*, 2006.
10. J. Zdziarski. *Ending Spam.* No Starch Press, 2005.

Appendix: Charts

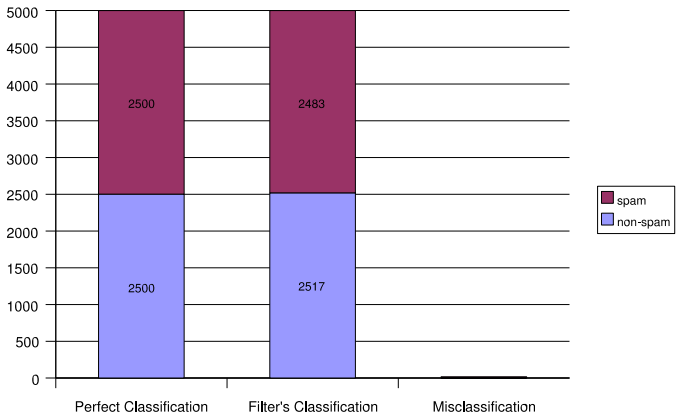


Fig. 2. Test Case 1

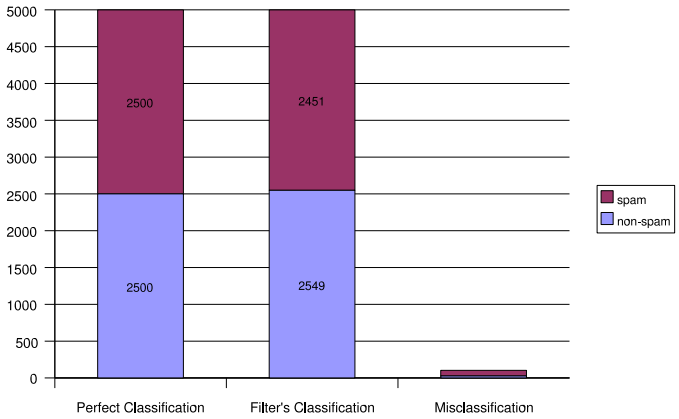


Fig. 3. Test Case 2

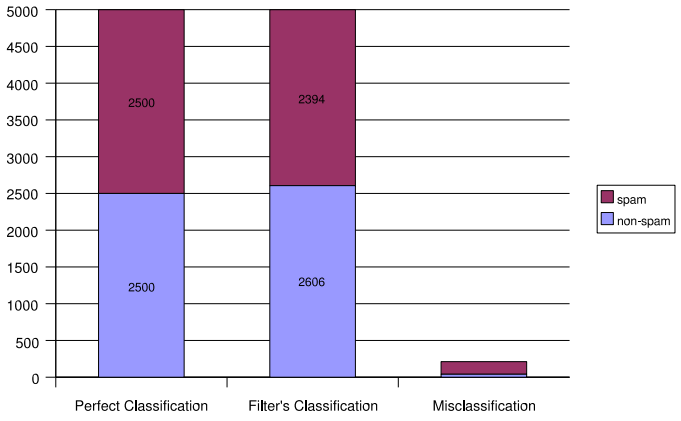


Fig. 4. Test Case 3

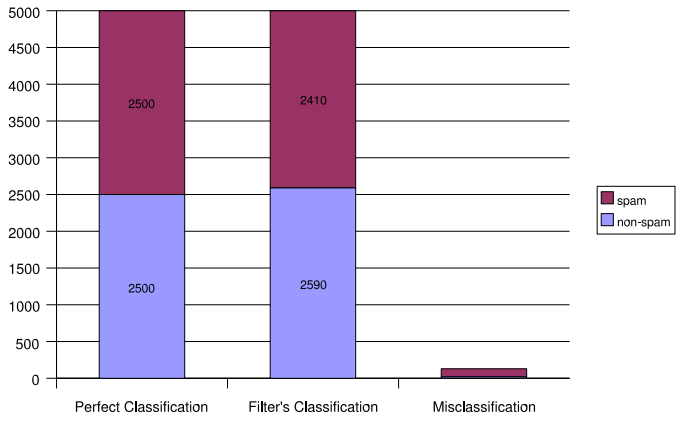


Fig. 5. Test Case 4