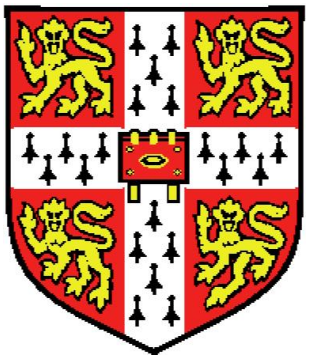
Spam Filtering with Naive Bayesian Classification



Khuong An Nguyen

Queens' College

University of Cambridge

L101: Machine Learning for Language Processing

MPhil in Advanced Computer Science

09-April-2011

Abstract

Spam (junk-email) identification is a well-documented research area. A good spam filter is not only judged by its accuracy in identifying spam, but also by its performance. This project aims to replicate a Naive Bayesian spam filter, as described in *'SpamCop: A Spam Classification & Organization Program'* paper. The accuracy and performance of the filter are examined on the GenSpam corpus. In addition, the project investigates the actual effect of the Porter Stemming algorithm on such filter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of Internet has popularised E-mail as an effective means to communicate between people. At the same time, it has encouraged a new form of advertising known as spam or junk-email. Sophisticated spammers forge their e-mail headers so that it can bypass many filters relying on address checking. In this project, the filter relies on the actual message content to distinguish spam emails. There are three distinct processes. First, the filter is supplied with many training data, including both genuine and spam e-mails. Second, the filter removes redundant words and smooths data by applying the Porter Stemming algorithm. Finally, the testing e-mails are passed into the filter for classification.

1.1 Report outline

This report is divided into five chapters.

Chapter 1 outlines the moviation of the spam filtering area, and the overview of a standard probabilistic filter.

Chapter 2 covers the backgrounds of the project, including Bayesian network, Naive Bayesian classifier and the Porter Stemming algorithm.

Chapter 3 constructs a break-down view of the spam filter implemented in this project, covering four transition states of the system from training

to actual deployment.

Chapter 4 continues with the spam filtering system, by examining the performance results on the GenSpam corpus, along with an in-depth analysis of the actual performances reported in the paper.

Chapter 5 concludes with a summary of contributions, programming issues, and an outline of scope for future work.

The logical progression of the project is graphically depicted in figure 1.1.

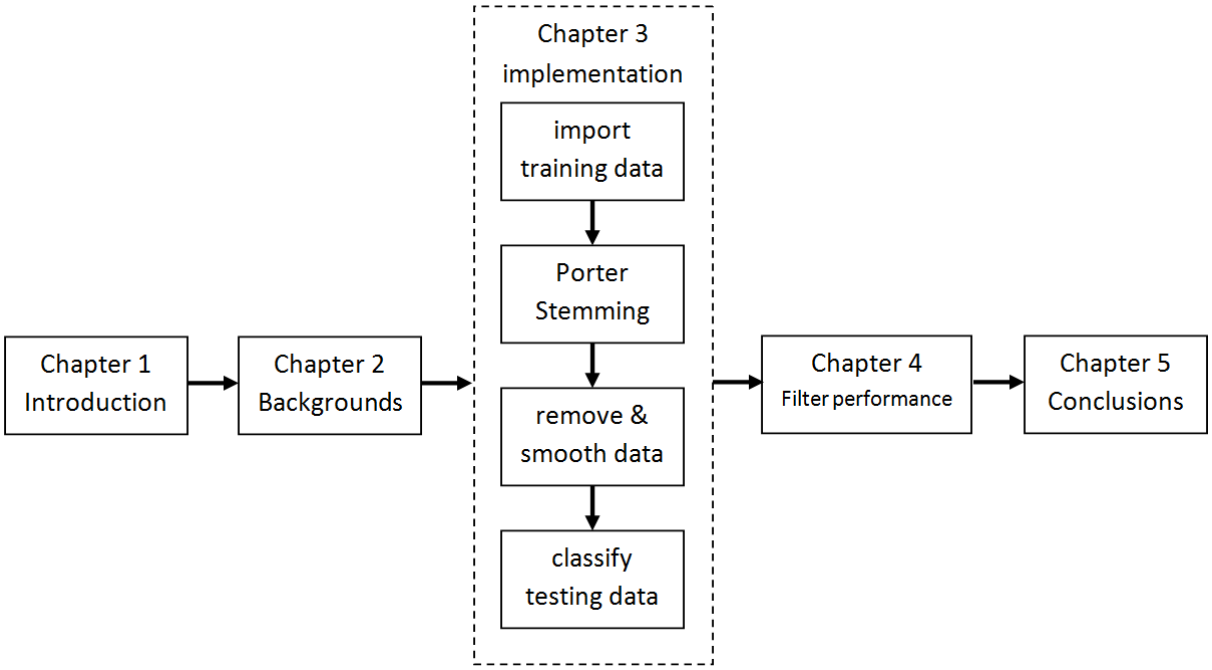


Figure 1.1: Report progression

Chapter 2

Naive Bayesian Classification and Porter Stemming algorithm

2.1 Bayesian classification

A Bayesian network is a directed acyclic graph, which represents a set of nodes and their dependencies. A directed edge from node X to node Y as seen in figure 2.1, means Y conditionaly depends on X.

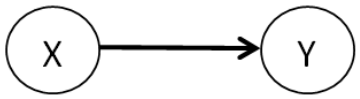


Figure 2.1: Conditional dependence

A node N is said to be conditionaly independent of node M if N does not directly connect with M. Each node X is assigned with a probability table, which specifies the distribution over X, given the value of X's parent.

Given a classification task, the Bayesian network can be applied to predict the decision outcome. For example, given the number of working hours and the stress level, the Bayesian network can represent the probabilistic relationship between number of working hours and the stress level. Then, if given a particular working hour number, the Bayesian classifier can work out the probability of the stress level.

The standard formulae of Bayesian classifier is

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{2.1}$$

P(X | Y) stands for the probability of the event X, given the event Y.

2.2 Naive Bayesian classification

Naive Bayesian classifier is simply the Bayesian classifier relaxed the dependency assumption. In particular, Naive Bayesian assumes that the presence or absence of any node in the Bayesian network does not affect any other nodes. For example, considering 'wet grass' and 'cloudy' as the two nodes of the network, although they both contribute to the probability of 'raining' event, the existence of 'wet grass' event does not affect the existence of 'cloudy' event and vice-versa.

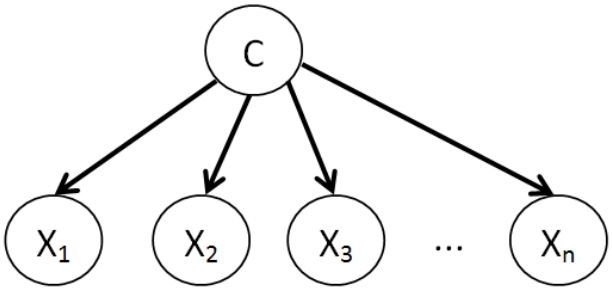


Figure 2.2: Conditionally independent

The biggest advantage of Naive Bayes is the computational overhead reduction, in order to estimate a probability. The quantity P(X | Y) in the formulae (2.1) is often impractical to calculate directly, without any independence assumptions. Since each node $x_1, x_2, \ldots x_n$ are conditional independent of each others, given a common class C, as depicted in figure 2.2, its probability can be calculated separately, and the combination of separate probability of each node can be combined to yield an overall probability of the big event. The general formulae for Naive Bayesian in terms of each separate node can be calculated as:

$$P(X|C) = P(x_1|C)P(x_2|C)\ldots P(x_n|C) \tag{2.2}$$

2.3 Porter Stemming algorithm

The Porter Stemming algorithm is widely used in the Information Retrieval area, due to the vast amount of data extracted from a document. The purpose of Porter Stemming is to group similar words, hence improving performance because the extracted data volume is reduced.

The algorithm takes any English word and returns its root form, by removing the word's suffix. For example, all three words 'teacher', 'teachers', 'teaching' convey the same base idea, and will be reduced to a shorter form 'teach' by the algorithm. Before examining details, it is worth noting that the algorithm is very careful not to remove any suffix when the word is already too short, to avoid losing the meaning of the word. Also the final result of the returned word is not necessarily meaningful in the linguistic form.

First, the definitions of 'vowel' and 'consonant' need to be clarified, as they are intensively used by the algorithm. A vowel in English is the letter A, E, I, O, or U. If the letter Y is not preceded by a vowel, it is considered as another vowel. Any of the letters which are not vowels are called consonant. For example, considering the word 'holy', the letter 'h' and 'l' are vowels, while 'o' and 'y' are consonant.

If a group of continuous vowels are denoted as V, and a group of continuous consonants are denoted as C, any English word will have one of the four following forms:

CVCV ... C
CVCV ... V
VCVC ... C
VCVC ... V

From the four above forms, a single united form can be defined as $[C](VC)^m[V]$, with [C] and [V] means C and V can be absent or present. For example:

with m = 0, an acceptable word is 'tree'.
with m = 1, an acceptable word is 'play'.
with m = 2, an acceptable word is 'teachers'.

To remove a suffix from an English word, the Porter Stemming algorithm applies the following rule:

(condition) S1 $\rightarrow$ S2

This rule states that if the word satisfies the specified 'condition', and it also ends with the suffix S1, then S1 will be replaced with S2. Given a set of rules, the algorithm will prioritise the one with the longest matching S1. For example, given the following three rules:

(m = 2): ers $\rightarrow$ ing [1]
(m = 2): s $\rightarrow$ [2]
(m = 2): rs $\rightarrow$ s [3]

The word 'teachers' satisfies all three conditions, however the rule [1] is applied first since it has the longest matching 'ers' suffix. The word 'teachers' will be replaced as 'teaching'.

The algorithm has five steps, in which a complicated long word is stripped bit by bit in each step to its root form. Each step contains a set of rules which strip all common suffixes such as plural forms, passive form, noun, etc..

Chapter 3

Filter implementation

Having already discussed the Naive Bayesian classifier and the Porter Stemming algorithm, this chapter looks further to apply them in an actual spam filter. The chapter follows four transition states as the filtering system develops. First, the training data is imported to the system. The Porter Stemming algorithm is applied to every word in the training data to produce 'a token'. Then the new extracted tokens are passed through a small filter which removes any common words. Data is then stored in the training system and the final step simply queries the probability of each word required for classification.

3.1 Importing training data

This is the learning step, which teaches the filter which e-mail is spam and what is genuine. Two sets of training data are fed into the system. For each e-mail, the message content is broken down into smaller words. A word is a consecutive sequence of characters. Two words are separated by one or many spaces.

The training data is encoded into a probability table. This table stores the probability of every word found in the training data, categorised into Spam and Non-spam classes. Thus, for some training data, a particular word will have a different probability for the Spam class and a different probability for the Non-spam class. To generate such table, the system first counts the frequency of each individual word in each class. Then, the m-estimate function is applied to that

word to calculate the corresponding probability. The formulae (3.1) and (3.2) show how to calculate the probability of each word in the Spam and Non-spam class. The Freq(word) function calculates the number of times the word appears in the equivalent class, similarly Freq(all-words) returns the total size of the class. The constant K is the number of unique words in the whole training set.

$$P(word|Spam) = \frac{Freq(word) + \frac{1}{K}}{Freq(allwords) + 1} \tag{3.1}$$

$$P(word|Nonspam) = \frac{Freq(word) + \frac{1}{K}}{Freq(allwords) + 1} \tag{3.2}$$

3.2 Porter Stemming application

The Porter Stemming module receives a word and returns a new token in its original form, by stripping out the redundant suffixes, as discussed in chapter 2.3. By returning the words to their root form, the training data size can be greatly reduced, as well as increasing the probability of the word, hence resulting in a better classification.

The algorithm is applied by the system when any new word is presented. The training data only stores the processed tokens returned by the algorithm.

3.3 Removing and smoothing data

There are many 'stop words' in English. These are the most frequent words such as *'the', 'a', 'of'*. Their presence do not contribute to the primary meaning of the sentence in particular, or the whole message's content in general. However, they do occupy a significant amount of space as will be discussed in chapter 4. Not only do they increase the computational overhead, they also affect the final result caused by accumulating small errors. The opposite of those 'stop words' are the 'rare words'. These words appear so rarely in the training data and do not help the final result either.

For this module, the system applies a small filter, which removes all 'stop words' and 'rare words' from the training data. In particular, the system removes all 'rare words' appearing fewer than four times within all spam and non-spam training e-mails. The system also removes all 'stop words' mentioned above, when their occurence is more than 10% in the whole training data.

3.4 Classifying a new email

When a new email which needs to be classified, is presented to the filtering system, it is broken down into smaller words. The Porter Stemming algorithm is applied to these words. A question arises here: what if the original words are passed to the system without applying the Porter Stemming. The answer is the probability of the word will be extremely small, because the original form of the word is not recorded in the training data. In this case, these words do not contribute to the overall probability of the whole message.

The message processing in this step is similar to the one at the importing training data in step 1. However, to improve the system running time, the small filtering process to remove the most frequent words and rare words is ignored. This does not affect the results at all, because the system returns an extremely small probability for any frequent words appearing as they are not found in the training data.

By putting all processed words together, every e-mail can be presented as a vector $(a_1, a_2, \ldots, a_n)$, with n is the total number of tokens in an e-mail. According to the Bayesian network formulae, given two class labels SPAM and NONSPAM

$$P(Spam|(a_1, a_2, \ldots, a_n)) = \frac{P((a_1, a_2, \ldots, a_n)|Spam)P(Spam)}{P((a_1, a_2, \ldots, a_n))} \quad (3.3)$$

$$P(Nonspam|(a_1, \ldots, a_n)) = \frac{P((a_1, \ldots, a_n)|Nonspam)P(Nonspam)}{P((a_1, a_2, \ldots, a_n))} \quad (3.4)$$

$P(Spam|(a_1, a_2, \ldots, a_n))$ states the probability this e-mail spam, given a vector $(a_1, a_2, \ldots, a_n)$, similarly to $P(Nonspam|(a_1, a_2, \ldots, a_n))$. To decide if an e-

mail is spam or genuine, the system compares the two terms. If $P(Spam|(a_1, a_2, \ldots, a_n)) > P(Nonspam|(a_1, a_2, \ldots, a_n))$, the e-mail is identified as spam and vice-versa. Besides, as P(Spam), P(Nonspam), and $P((a_1, a_2, \ldots, a_n))$ are constants, the problem becomes finding which one is larger between $P((a_1, a_2, \ldots, a_n)|Spam)$ and $P((a_1, a_2, \ldots, a_n)|Nonspam)$.

Since each word in the e-mail is conditionally independent, given the class label SPAM or NONSPAM, the Naive Bayesian classifier says that:

$$P((a_1, a_2, \ldots, a_n)|Spam) = P(a_1|Spam)P(a_2|Spam)\ldots P(a_n|Spam) \quad (3.5)$$

$$P((a_1, a_2, \ldots, a_n)|Nonspam) = P(a_1|Nonspam)\ldots P(a_n|Nonspam) \quad (3.6)$$

Each small term $P(a_i|Spam)$ and $P(a_i|Nonspam)$ can be obtained directly from the probability table as discussed in chapter 3.1. When the training data is not sufficient, a word's probability might not be found in the table. It is very important that this word gets a small probability such as 0.0000001 to prevent the whole equation turning into zero.

Chapter 4

Experimental results

A good spam filter is judged on its safety measured by the percentage of genuine messages which are incorrectly classified as spam, and the efficiency measured by the percentage of spam e-mails passing un-noticed by the filter. In this chapter, the system is tested with the GenSpam corpus. The success of the system will be justified sorely on the experimental results in terms of executing time, safety and efficiency. An in-depth analysis of the contribution of the Porter Stemming algorithm is also documented.

4.1 GenSpam corpus

The GenSpam corpus used in this project consists of:

9,072 genuine e-mails with 154,000 words.

32,332 spam e-mails with 281,000 words.

In particular, the above e-mails are divided into

8,018 genuine e-mails + 31,235 spam e-mails for the training set.

754 genuine e-mails + 797 spam e-mails for the testing set.

300 genuine e-mails + 300 spam e-mails for the adaption set.

The imbalance of the ratio between genuine e-mails and spam e-mails can be explained that spam e-mails are considerably shorter than the genuine e-mails. All messages are collected within the same period of times between 2002-2003. To preserve information confidentiality, five tokens are used to anonymise sensitive

information

&NAME: person name

&CHAR: individual characters

&NUM: numbers

&EMAIL: e-mail address

&URL : internet url

The e-mail content is presented in XML format, with mark-ups tags such as <FROM>, <TO>, <MESSAGE>, etc ... After the anonymisation procedure, every single e-mail is manually examined to anonymise any remaining sensitive information. Although most useful information is lost during these processes, the corpus' author confirmed that the performance is only slightly worse, and is not sufficient enough to cause any misclassification.

Upon processing the corpus' data, there are some small issues taken into consideration, such as, there is a <space> character at the end of <TEXT_NORMAL>, <TEXT_EMBEDDED>, <MESSAGE> tags in certain lines, which may causes the system to mis-process the line's content. Besides, there are very long words such as one with 567 characters at line 80,839 in train_SPAM.ems file.

4.2 Filter performance

The filter's performance will be judged on three criteria: the safety, the efficiency and the running time. The safety is the percentage of genuine e-mails that are correctly classified as genuine by the filter. The efficiency is the percentage of spam e-mails classified as spam by the filter. And the running time is the total time needed from training to classifying data.

In general, the safety of the filter is the most important feature, because it would be a disaster to discard an important genuine e-mail. However, letting some spam e-mails passing through would be acceptable in many cases. The training time can be tolerant, as long as the classifying time is not too bad.

To make a better judgement of the system, this section is divided into three analysis. First, the overall performance of the two testing sets and the two adaptation sets is discussed. Next, the size of training data is considered, to measure how well the system performs under different inputs. Finally, the actual effect of

the Porter Stemming is investigated. At the same time, a comparison with the actual results reported in the paper is also documented.

4.2.1 Overall system performance

Table 4.1 summarises the performance details on the two sets of data, the testing-set contains 754 genuine e-mails and 797 spam e-mails, while the adaptation-set contains fewer e-mails with just 300 genuine e-mails and 300 spam e-mails. Both sets are taken from two users' inboxes. The system was trained with 8,018 genuine e-mails and 31,235 spam e-mails.

Table 4.1: Overall system performance

Data to classify	Safety	Effectiveness	Running time(minutes)
Test-set	96.2%	94.98%	5.7
Adaptation-set	92.67%	90.67%	2.1

The result reported in the SpamCop paper that the filter is safe up to 98.84% is questionable, according to the actual performance on the GenSpam corpus. This number is even lower at 91.67% for the adaptation-set. However, the efficiency measured is actually higher than what reported in the paper at 92% for the testing-set.

Table 4.2 collects three randomly selected genuine e-mails classified as spam by the filter in the testing-set. The first two emails are simply testing e-mails without any important information. The third e-mail contains some popular words amongst spam e-mails such as *'sexuall', 'junk', 'hunk', 'CLICK HERE'* which can be understandable to be classified as spam by the filter. Overall, for all 3.8% of the genuine e-mails classified as spam by the filter, most of them are not too important.

4.2.2 Training data size

The difference from what reported in the paper can be explained by the difference in training data size, since the paper's filter was only trained with 160 spams and

Table 4.2: Genuine e-mails classified as spams

E-mail no.	Message content
1	&NAME EMAIL A test .
2	&NAME EMAIL &NAME , A test . &NAME
3	Re : &NAME : &NAME Deals on what you ordered Absolutely ! Was there an image with it ? ? &CHAR On &NAME , &NUM May &NUM , &NAME &NAME wrote : is this the kind of hunk male (I mean junk mail) you mean ? ? ? :-O Dont let a ' SMALL ' problem be your down fall ! Feel better look better even make your sexuall partner happy ! CLICK HERE To be removed from out opt-in list Click here

466 non-spam e-mails, while the system in this project was trained with 31,235 spams and 8,018 non-spams e-mails from the GenSpam corpus.

To verify this assumption, the test-set in the GenSpam corpus will be used as training data, which has much fewer data, and the system is tested against the adaptation-set, which has similar size as the testing data used in the paper. Surprisingly, the result from table 4.3 perfectly matches what reported in the paper, with the safety and the efficiency at 98.33% and 91.33% respectively. A similar result was obtained when the adaptation-set is used as training data and the test-set is used as testing data.

Table 4.3: Overall system performance with fewer training data

Training data	Data to classify	Safety	Effectiveness	Running time(minutes)
Test-set	Adaptation-set	98.33%	91.33%	3.1
Adaptation-set	Test-set	97.21%	88.46%	3.5

4.2.3 Porter Stemming effect

Having known the effect of Porter Stemming is to group relevant words and to reduce the computational overhead, this part investigates the filter performance without applying the 'Porter Stemming'. The two tests above are repeated but the 'Porter Stemming module' is switched off. An expectation would be a lower safety, effectiveness, and a longer running time.

Table 4.4 re-assures the above expectation, although it is surprising not to see much influence (around 4%-5% in most the case) when the 'Porter Stemming' is switched off. The running time is just a little bit slower too.

Table 4.4: System performance without 'Porter Stemming'

Training data	Data to classify	Safety	Effectiveness	Running time(minutes)
Training-set	Test-set	92.41%	89.72%	7.3
Training-set	Adaptation-set	87.2%	88.06%	3.2
Test-set	Adaptation-set	94.7%	87.96%	4.6
Adaptation-set	Test-set	92.18%	85.16%	5.3

To understand how much computations the Porter Stemming can reduce, figure 4.1 spots the amount of words effectively removed by the algorithm for three data sets. It can be said that the bigger the size of data is, the more effective Porter Stem algorithm is. In particular, for the training set with 90,391 words originally, the total amount of words is reduced to 76,676 after stemming.

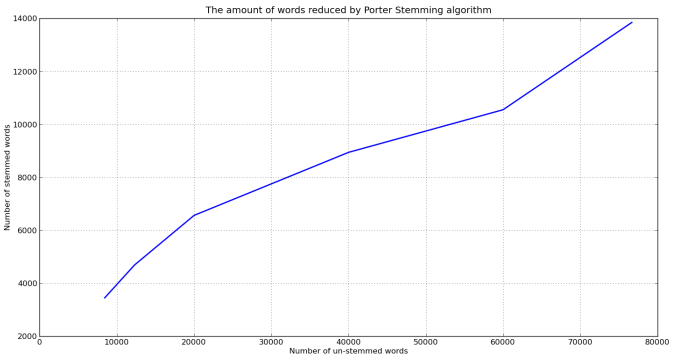


Figure 4.1: The amount of words reduced by Porter Stemming

Chapter 5

Conclusions

In this project, we implement a spam filtering system using Naive Bayesian classification with the Porter Stemming algorithm. Besides, we perform an in-depth analysis of the filter performance in terms of safety, efficiency and running time. The actual effect of Porter Stemming on the system performance is also investigated.

In general, all above goals are achieved. In this chapter, we present other practical programming issues facing through-out this project, as well as an outline of the scope for future work.

5.1 Programming issues

Our system is programmed in Java. We programmed the whole Bayesian classification from scratch without the help of external tools. During the programming process, there are several problems in which we encountered and led us to improve our system.

One of the biggest problems which is not mentioned in any papers is the arithmetic calculation problem. Since the probability of each word in the training data ranges from zero to one, plus the amount of words in each e-mail is very large, by multiplying these probabilities many times to calculate the Naive Bayesian product, we get an extremely small number, with more than seven hundred digits after the decimal point. We prevented rounding up these numbers to preserve the precision. However, Java language cannot contain such extreme precision,

and at some point, the number is returned as ZERO. To fix the problem, we use a monotonic function $y = 10^{30}x$. When the probability of a given word gets too small, the function is applied to the probability of both SPAM and NONSPAM classes. This approach got its own issue. If the probability of the SPAM class keeps getting too small, while the probability of the NONSPAM class stays in an acceptable range, since the function must be applied to both classes at the same time to maintain the ratio, the probability of NONSPAM will go to INFINITY at some point, thus we use another monotonic function $y = 10^{-10}x$ to keep both probabilities in the correct decimal range.

5.2 Future work

Instead of using the Porter Stemming algorithm, other techniques in the Information Retrieval area could be applied to reduce the size of the vector space representing an e-mail. Another replacement would be the lemmatization in the RASP system.

Besides, the training data could be hashed, so that any word's probability can be quickly returned. At the moment, we have to search through every single word in the training data, whenever a word's probability is requested.

Appendix

All ready-to-run the Java source codes used in this project can be downloaded at: **http://khuong.vn/L101/ProjectCode.zip**
The GenSpam corpus can be downloaded at: **http://www.cl.cam.ac.uk/Research/NL/nl-download/GenSpam.tar.gz**
All results obtained in this report can be re-produce with the provided codes.

References

[1] P. Pantel, D. Lin: *"SpamCop: A Spam Classification & Organization Program"* (1998).

[2] B. Medlock: *"An Adaptive, Semi-Structured Language Model Approach to Spam Filtering on a New Corpus"* (2006).

[3] M. Sahami, S. Dumais, D. Heckerman, E. Horvitz: *"A Bayesian Approach to Filtering Junk E-Mail"* (1998).

[4] W. Frakes, R. Baeza-Yates: *"Information Retrieval: Data Structures & Algorithms"* (1992).