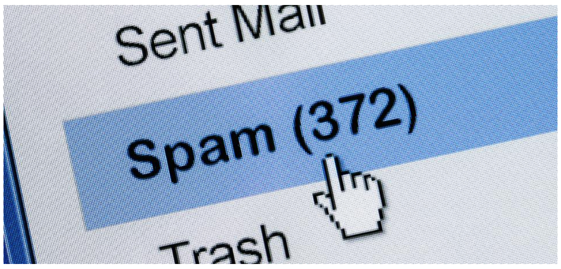
Random modelling project
Spam filtering in email sorting

Fadoua Bousalim, Antoine Haie, Joséphine Izard, Charline Le Lan, Moises Botarro Ferraz Silva

July 2017

Abstract

We study and implement a first approach to spam email filtering, based on Naive Bayes classification. Considering the naive assumptions such a filter is built on (word independence, positional independence), we illustrate its limits - using text generation - and try to understand its paradoxical effectiveness.



Contents

1 Context

Spam refers to unsolicited online messages - especially advertising posted on data networks or sent as email. Spam emails clutter mailboxes : as a consequence, "legitimate" (non-spam) emails may be delayed, people delete spams with the danger of deleting legitimate emails, a bandwidth must be allowed for people with a dial-up Internet connection to download junk emails, etc.

Spam filtering is a particular instance of the text categorization problem - assigning predefined categories to free-text documents - using two classes : S (spam) and L (legitimate, also referred to as *ham*). Our goal is to find the best class for any incoming email d.

In statistical spam filters, the identification of spam features is dynamic (see [1]). While other spam filters require the author to program the characteristics of spam, those filters automatically select the features of spam and non-spam and calculate the probability of an incoming email being spam.

2 Statistical spam filtering

2.1 Naive Bayes spam filtering

Naive Bayes filtering is based on the idea that words have different probabilities of occurring in spam emails and legitimate emails.

Based on single words occurrences We consider a dictionary, whose size we fix. Let $\langle t_1, ..., t_{n_d} \rangle$ be the sequence of terms in email d that belong to the dictionary we use for classification, as it occurs in d. We are looking for the probability that the email belongs to a class $c \in \{S, L\}$ given the occurrence of those dictionary words : $P(c|d) = P(c|\langle t_1, ..., t_{n_d} \rangle)$.

Naive word independence assumption To reduce the number of parameters, we make the Naive Bayes conditional independence assumption : we assume that words in an email are statistically independent[1].

$$P(\langle t_1, ..., t_{n_d} \rangle|c) = P(X_{k_1} = t_1, ..., X_{k_{n_d}} = t_{n_d}|c) = \prod_{1 \le k \le n_d} P(X_k = t_k|c),$$

where X_k is the random variable giving the dictionary term in position k in an email d. $P(X_k = t|c)$ is the probability that term t will occur in position k in an email d (see [2]).

Positional independence assumption Considering a different probability distribution for each position k involves too many parameters. As the position of a term in an email d does not help much to determine if it d is spam or non-spam, we make a second independence assumption : the conditional probabilities for a term are independent of position in the email (see [2]). For all positions k_1, k_2, for all terms t :

$$\forall c \in \{S, L\}, P(X_{k_1} = t|c) = P(X_{k_2} = t|c).$$

[1]In fact, words are not chosen randomly but with very significant correlation. Nevertheless, Bayesian discrimination works surprisingly well, as we will see.

Bayes' chain rule Let $P(c)$ denote the prior probability of an email occurring in class c. Let $P(t_k|c)$ denote the conditional probability of term t_k occurring in an email of class c. It is interpreted as a measure of how much evidence t_k contributes that c is the right class[2]. The naive word independence assumption gives:

$$\forall k \in \{1, 2, ..., n_d\}, P(t_k|c, t_1, ..., t_{k-1}, t_{k+1}, ..., t_{n_d}) = P(t_k|c).$$

Using Bayes' chain rule, the probability of an email d being in a class $c \in \{S, L\}$ is computed as :

$$P(c|d) = P(c|t_1, ..., t_{n_d}) = \frac{P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)}{P(t_1, ..., t_{n_d})}.$$

Let $\hat{c}$ denote the class returned by our Naive Bayes classifier :

$$\hat{c} = \arg \max_{c \in \{S,L\}} \hat{P}(c|d) = \arg \max_{c \in \{S,L\}} \hat{P}(c|t_1, ..., t_{n_d})$$

where $\hat{P}$ denotes an estimated probability (see [3]). We dropped the term $P(t_1, ..., t_{n_d})$ because is the same for both classes (S spam and L legitimate) and does not affect the arg max. Hence, we use the following classification rule:

$$\hat{c} = \arg \max_{c \in \{S,L\}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c) \tag{1}$$

where $\hat{P}(c)$ and $\hat{P}(t_k|c)$ denote the estimations of $P(c)$ and $P(t_k|c)$ obtained from a training data set (the probabilities and decision criterion of our filter are learned automatically from training data).

Filter training To estimate parameters $\hat{P}(c)$ and $\hat{P}(t|c)$, we train our Bayesian filter over a set of emails pre-classified as spam or legitimate. We use *maximum likelihood estimates*, i.e. relative frequencies which give the most likely value given the training data set (see [3]).

- The prior probability that a random email belongs to class c, $P(c)$, is typically estimated by:

  $$\hat{P}(c) = \frac{N_c}{N},$$

  where N_c denotes the number of emails from class c in the training data set and N denotes the total number of emails in the training data set.

- The conditional probability of term t occurring in an email of class c, $P(t|c)$, is estimated by:

  $$\hat{P}(t|c) = \frac{T_c^t}{\sum_{t'} T_c^{t'}},$$

  where T_c^t is the number of occurrences of t in the training data set from class c (including multiple occurrences in an email). Each word feature t generates one such probability.

[2]In the case of an email whose terms do not provide clear evidence for one class or the other, we choose the class c that has the higher prior probability $P(c)$.

Smoothing The previous formula is problematic: if a feature t does not appear in class c in the training set (i.e. $T_c^t = 0$), then $\hat{P}(t|c)$ is equal to 0. For instance, if the dictionary word "modelling" only occurred in spam emails in the training data set, then an email which contains the sentence "This is a random modelling project on a method of spam filtering" will get a conditional probability for class "legitimate", $\hat{P}(L|d)$, of zero (1).
We want to implement classifiers which prevent zero probabilities in further computations. A "Laplace smoothing" simply adds 1 to each count ($T_c^t \to T_c^t + 1$). The estimate of the probability for the occurrence of the term t in class c is given by :

$$\hat{P}(t|c) = \frac{(T_c^t + 1)}{\sum_{t'} (T_c^{t'} + 1)} = \frac{T_c^t + 1}{\sum_{t'} T_c^{t'} + n}$$

where n is the number of word features (i.e. terms in the dictionary).

Weighting schemes A critical step is to choose the email representation. In our code, we choose to represent an email d by the vector $\vec{x} = (x_1, x_2, \ldots, x_n)$, where x_k is the number of occurrences of dictionary term t_k in d. There are various Naive Bayes classifiers, corresponding to different weighting schemes (see [4] and [5]).

- Multinomial Naive Bayes classifier:
  In the multinomial Naive Bayes model, each email d from class c is seen as the result of picking independently $|d|$ terms from dictionary $(t_1, ..., t_n)$ with replacement, with probability $P(t|c)$ for each t. Then, $P(\vec{x}|c)$ follows the multinomial distribution:

  $$P(\vec{x}|c) = P(|d|)|d|! \prod_{1 \le k \le n} \frac{\hat{P}(t_k|c)^{x_k}}{x_k!}$$

  where we have followed the assumption that $|d|$ does not depend on the class[3].

- Gaussian Naive Bayes classifier:
  The gaussian Naive Bayes model assumes that each word feature follows a normal distribution $g(x_k; \mu_{k,c}, \sigma_{k,c})$ in each class c:

  $$g(x_k; \mu_{k,c}, \sigma_{k,c}) = \frac{1}{\sqrt{2\pi\sigma_{k,c}^2}} \exp(-\frac{(x_k - \mu_{k,c})^2}{2\sigma_{k,c}^2})$$

  where parameters $\sigma_{k,c}$ and $\mu_{k,c}$ are estimated using *maximum likelihood.*

  $$P(\vec{x}|c) = \prod_{1 \le k \le n} g(x_k; \mu_k, \sigma_k)$$

Naively, if $P(S|\vec{x}) > P(L|\vec{x})$, the email represented by x is classified as a spam. Else, it is classified as a legitimate email. However, it is worth noticing that classifying a legitimate email as spam (false positive) is worse than letting a spam get through the filter (false negative). Let λ denote the cost (meaning the risk) of misclassifying an email of class L as belonging to class S. Taking $\lambda = 1000$ means that classifying a legitimate email as spam is as bad as having 1000 spams getting through the filter. We may thus rewrite the classification rule $\frac{P(S|x)}{P(L|x)} > \lambda$: the email represented by $\vec{x}$ is moved to the spam folder if the ratio of the probability that it is unwanted to the probability that it is legitimate is greater than λ (see [6]).

This method has a learning ability, although slender, since the filter must be trained to build up probabilities $P(\vec{x}|c)$ that the filter does not know in advance. The need for manual classification is not eliminated: the training emails have been labeled (as spam or legitimate) by the user.

[3]Questionable in spam filtering : for instance, the probability of a very long spam email might be smaller than that of an equally long legitimate email.

2.2 Code and analysis of performance

We implemented a naive Bayes email classifier using Python© ("Part 1 - spam filtering.py"). The data set used to train and test our filter is a set of emails extracted from the "Ling-spam", a publicly available collection of spam and non-spam emails (can be downloaded from [7]). We walked through the following steps :

Finding reprocessed data Our data set is split into a training data set and a testing data set (respectively 702 mails in "train-mails" folder and 260 mails in "test-mails" folder), each divided equally between spam and ham mails. The data set we use is reprocessed : 1) it underwent the removal of certain characters - punctuation marks, digits, extremely common words like "the", "and", "of" - regarded as irrelevant in spam detection ; 2) it underwent lemmatization to reduce inflectional forms and create a common base form - for instance, "modelling" and "modelling," are reduced to a common base form "modelling".

Creating a dictionary using the training data set Our delimiter is a space – separator in most languages. Thanks to our reprocessed data, we can avoid considering auxiliary delimiters to solve punctuation issues (e.g. separating features such as "modelling" and "modelling,"). We consider the body of each email. We implement a function called WORDS_IN_DICTIONARY, which depends on the training data set and the size of the dictionary desired, and creates a dictionary of chosen words and their counts.

Extracting word features from the testing data set Our function MATRIX_OF_OCCURRENCES returns the occurrences of word features (i.e. words in the dictionary) in the testing data set : it generates a matrix whose i, j component gives the number of occurrences of the j^{th} word feature in the i^{th} email tested.

Training the spam filter We used the scikit-learn package for Python© to train our Bayesian filters with weighting schemes MULTINOMIALNB() and GAUSSIANNB() (see [4]).

Assessing the performance We assess our filters' performance on a TESTING_ARRAY which gives the known labels spam/ham for 260 emails tested. We compare them with our filter's classification using MATRIX_OF_OCCURRENCES and PREDICT - which performs classification and predicts the class (spam/ham) of the emails tested. We treated spam filtering as a binary classification problem for detecting spam : a spam email can be considered as a positive (P) instance and a legitimate email (or ham) as a negative (N) instance. CONFUSION_MATRIX returns our spam filters' confusion matrix. Its diagonal elements number correctly classified emails ("true positive", "true negative") whereas its non-diagonal elements number wrongly classified ones ("false positive", "false negative").

Weighting scheme	Ham	Spam
Ham	TN	FN
Spam	FP	TP

We define the accuracy and the false alarm rate of our classifiers as the ratio of the number of correctly classified emails to the number of emails tested and the ratio of the number of legitimate emails considered as spams to the number of legitimate emails:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} = \frac{\text{TP} + \text{TN}}{260} \qquad \text{False Alarm Rate} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{130}$$

3000-words dictionary :
When we allow the dictionary to contain 3000-words, the filter implemented with the MultinomialNB() weighting scheme correctly identifies 116 legitimate emails and 118 spams, and wrongly classifies 14 spams and - most importantly - 12 legitimate emails.

Multinomial NB	Ham	Spam
Ham	116	14
Spam	12	118

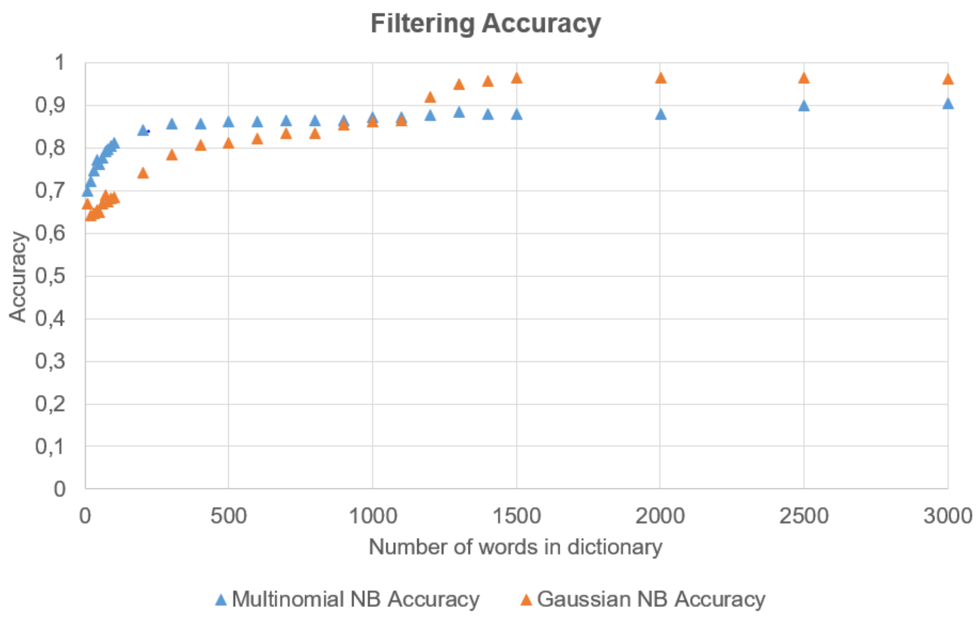
100-words dictionary :

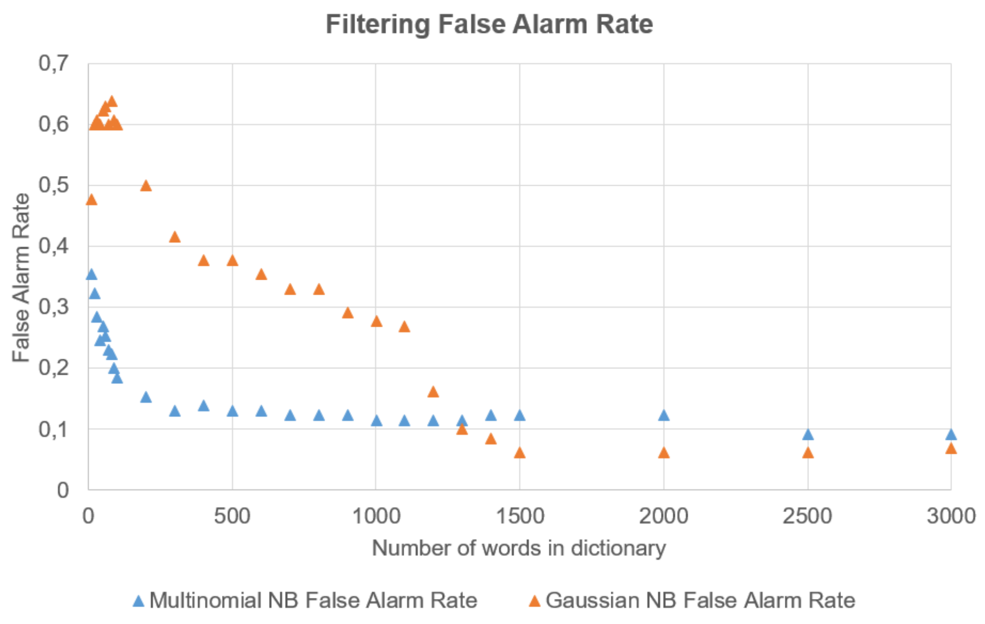
Multinomial NB	Ham	Spam
Ham	105	25
Spam	24	106

10-words dictionary :

Multinomial NB	Ham	Spam
Ham	98	32
Spam	46	84

Comparing Naive Bayes classifiers (Multinomial and Gaussian) Multinomial matching outperforms Gaussian matching in our case up to a certain size of dictionary, after which the Gaussian filter provides higher levels of accuracy. For small dictionaries, our spam filters' accuracy and false alarm rate depend heavily on the size of the dictionary n. Those performance indicators seem to reach a plateau at n_{max} - whose value depends on the Naive Bayes method (n_{max}(multinomial) $\simeq$ 2500 and n_{max}(gaussian) $\simeq$ 1500).





2.3 A surprisingly good spam classifier

With a 2500-words dictionary created from the training data set, we get accuracy rates of 90% and 97%, false alarm rates of $0,9\%$ and $0,6\%$ for the multinomial and gaussian weighting schemes respectively.

Our Naive Bayesian classification makes a *surprisingly* good spam filter. Notably, the word independence assumption it is built on does not hold for emails : in human language, words are conditionally dependent on each other...

3 Statistical text generation

Our Naive Bayes filter is so called because it relies on "naive" independence assumptions.

1. The conditional independence assumption states that, given a class c, word features are independent of each other, which is hardly true for words in an email: words "random" and "modelling" could be considered as highly dependent terms (not to mention "Hong" and "Kong").

2. Our model makes an assumption of positional independence: it discards positional information.

To illustrate the limits of such assumptions for human language, we can consider the corresponding text generation.

3.1 A bad text generator

To generate a text with our multinomial Naive Bayes model, we would first pick a class c with probability $P(c)$. Next we would generate term t_k in position k with $P(X_k = t_k|c)$ for each of the n_d positions for the text, the X_k having the same distribution over terms for a given class c.

The fact that an email contains the term "random" would not make it more likely or less likely that it also contains "modelling". This would result in a meaningless word generator rather than a real text generator.

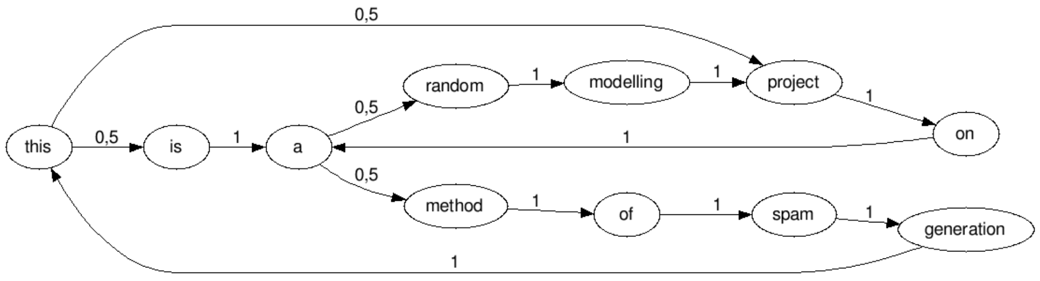
3.2 Markov chains and text generation

Some intelligible chains of words closer to human language can be built using the concept of Markov chains. We can implement a process to generate text based on Markov property.

We can represent every piece of training text as a Markov chain, making a list of the words it contains, their occurrences in the training text and the following word(s). Let us take the sentence "This is a random modelling project on a method of spam generation. This project." as the training data set.

```
1.  this      -> (1, is)(1, project)
2.  is        -> (1, a)
3.  a         -> (1, random)(1, method)
4.  random    -> (1, modelling)
5.  modelling -> (1, project)
6.  project   -> (1, on)
7.  on        -> (1, a)
8.  method    -> (1, of)
9.  of        -> (1, spam)
10. spam      -> (1, generation)
11. generation-> (1, this)
```

Following the word "this", there is a 50% chance the next word will be "is" and a 50% chance it will be "project". For the rest of the dictionary available, the following word is known for certain. It gives the following Markov chain:



3.3 Code and text generation

We implemented and trained such a text generator with a larger training text ("Part 2 - Markov text generation.py"). The generator goes through the following steps:

Choosing randomly a seed word We choose a word in the training text, preferably one that has many successors.

Fixing the successors We look for the row of the seed word in the transition matrix corresponding to the Markov chain and choose the next word randomly, according to the probabilities of the successors (probabilities in the same row). We repeat this step taking the next word obtained as the new seed word and generate the string of words. If at one point the seed word has no successor (rare for big training texts), the next word is chosen randomly.

The strings of words generated through this process - though not always grammatical and meaningful - statistically *resemble* the training data. The size of the training base has a tremendous impact on the "quality" of the text generated: the larger the better.

Given a training data set (input text), our Python© code ("Part 2 - Markov text generation.py") generates strings of terms (output text) using the previous method. Here is a 3-line text we generated ("result.txt") using an extract of Harper Lee's *To Kill A Mockingbird* (Chapter 21) as an input file ("text.txt").

```
Text generated:

have to return one verdict, but if you believe this, you’ll have to go home the
short way, because he walked quickly down the aisle.  ‘Miss Jean Louise?"  I looked
around. They were standing. All around us with biblical patience.  The old courthouse
```

Markov generated chains of terms are much closer to meaningful language than Naive Bayes generated texts.

4 How can our naive bayesian filter be a good spam classifier?

How can our Naive Bayes filter be an efficient email classifier when its model of human language is so oversimplified?

Although the *probability* estimates of our spam filter are questionable, its *classification decisions* are mostly correct.

Illustration We apply our multinomial spam filter (2500-words dictionary) to one of our reprocessed spam emails ("spsmgc81" in "spam" folder). PREDICT_PROBA returns the conditional probability estimates for each class (L in first column, S in second column). We get the following results ("Part 3 - spam filtering example.py"):

Class c	L	S
$\hat{P}(c)\prod_{1\leq k\leq n_d}\hat{P}(t_k\|c)$	0.00491447387451	0.493688947848
$\hat{P}(c\|d)$	0.00985647843636	0.990143521564

using $\hat{P}(c|d) = \frac{\hat{P}(c)\prod_{1\leq k\leq n_d}\hat{P}(t_k|c)}{\hat{P}(L)\prod_{1\leq k\leq n_d}\hat{P}(t_k|L)+\hat{P}(S)\prod_{1\leq k\leq n_d}\hat{P}(t_k|S))}$.

In fact, we notice for most emails tested that if c_1 is the class returned (and c_2 the other class), $\hat{P}(c_1|d)$ is *much bigger* than $\hat{P}(c_2|d)$. For an email d whose true conditional probabilities verify $\frac{P(S|d)}{P(L|d)} > 1$ (L for "legitimate", S for "spam"), containing positive indicators for the "spam" class and negative indicators for the "legitimate" class, our multinomial Naive Bayes classification frequently gives $\frac{\hat{P}(S|d)}{\hat{P}(L|d)} \gg 1$. The class returned ("spam") has a much bigger probability than the other class ("legitimate"). Hence, the estimates easily *diverge* from the true conditional probabilities ($\hat{P}(c|d) \simeq 0$ or 1, unlike most true $P(c|d)$) and eventually classify correctly the incoming email.

Improvement strategies In spite of their apparently over-simplified assumptions, Naive Bayes classifiers work quite well in many real-world situations, notably document classification and spam filtering. However, to bypass such a filter, spammers might incorporate in their spam email enough "ham" words to counterbalance the presence of other words clearly identified as spam indicators. The technique is called “dictionary salad” spamming.

A first improvement of our filtering strategy would be to consider not only single-word occurrences, but cliques of $k > 1$ words (Markovian filtering used in CRM114 discriminator, see [8] and [9]). We might also want to analyze IP addresses and particular HTML characters considered as good spam markers. Our filter could also dynamically adapt its initial training when false positives or false negatives are encountered (useful considering the evolving nature of spam).

References

[1] Zdziarski JA. *Ending Spam.* No Starch Press, 2005.

[2] https://nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html

[3] https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html

[4] http://scikit-learn.org/stable/modules/naive_bayes.html

[5] Rennie JD, Shih L, Teevan J, Karger DR. *Tackling the Poor Assumptions of Naive Bayes Text Classifiers.* In Proceedings of the Twentieth International Conference on Machine Learning, 2003.

[6] http://ats.cs.ut.ee/u/kt/hw/spam/spam.pdf

[7] http://aclweb.org/aclwiki/index.php?title=Spam_filtering_datasets

[8] Yerazunis WS. *Sparse Binary Polynomial Hashing and the CRM114 Discriminator.* In MIT Spam Conference, 2003.

[9] Yerazunis WS. *The Spam Filtering Accuracy Plateau at 99.9 Percent Accuracy and How to Get Past It.* In MIT Spam Conference, 2004.