On Using SVM and Kolmogorov Complexity for Spam Filtering

Sihem Belabbes[2] and **Gilles Richard**[1,2]
[1]British Institute of Technology and E-commerce,
Avecina House,258-262 Romford Road London E7 9HZ, UK
[2]Institut de Recherche en Informatique de Toulouse,
118 Rte de Narbonne 31062 Toulouse, France
grichard@bite.ac.uk
{belabbes,richard}@irit.fr

Abstract

As a side effect of e-marketing strategy the number of spam e-mails is rocketing, the time and cost needed to deal with spam as well. Spam filtering is one of the most difficult tasks among diverse kinds of text categorization, sad consequence of spammers dynamic efforts to escape filtering. In this paper, we investigate the use of Kolmogorov complexity theory as a backbone for spam filtering, avoiding the burden of text analysis, keywords and blacklists update. Exploiting the fact that we can estimate a message information content through compression techniques, we represent an e-mail as a multi-dimensional real vector and then we implement a support vector machine classifier to classify new incoming e-mails. The first results we get exhibit interesting accuracy rates and emphasize the relevance of our idea.

Introduction

Detecting and filtering spam e-mails face a number of complex challenges due to the dynamic and malicious nature of spam. A truly effective spam filter must block the maximum unwanted e-mails while minimizing the number of legitimate messages wrongly identified as spam (namely false positives). However individual users may not share the same view on what really a spam is.

A great deal of existing methods, generally rule-based, proceed by checking content of incoming e-mail looking for specific keywords (dictionary approach), and/or comparing with blacklists of hosts and domains known as issuing spam (see (Graham 2002) for a survey). In one case, the user can define his own dictionary thus adapting the filter to his own use. In the other, the blacklist needs to be regularly updated. Anyway, getting rid of spam remains a classification problem and it is quite natural to apply machine learning methods. The main idea on which they rely is to train a learner on a sample e-mails set (known as the witness or training set) clearly identified as spam or ham (legitimate e-mail), and then to use the system output as a classifier for next incoming e-mails.

Amongst the most successful approaches to deal with spam is the so-called Bayesian technique which is based on the probabilistic notion of Bayesian networks((Pearl & Russell 2003)), and has so far proved to be a powerful filtering tool. First described by P. Graham (Graham 2002), the Bayesian filters block over 90% of spam. Due to their statistical nature they are adaptive but can suffer statistical attacks simply by adding a huge number of relevant ham words in a spam message (see (Lowd & Meek 2005)). Next-generation spam filtering systems may thus depend on merging rule-based practices (like blacklists and dictionaries) and machine learning techniques.

A useful underlying classification notion is the mathematical concept of distance. Roughly speaking, this relies on the fact that when 2 objects are close one to another, it means that they share some similarity. In spam filtering settings, if an e-mail m is close to a junk e-mail, then m is likely a junk e-mail as well. Such a distance takes parameters like sender domain or specific words occurrences. An e-mail is then represented as a point in a vector space and spam are identified by simply implementing a distance-based classifier. At first glance an e-mail can be identified as spam by looking at its origin. We definitely think that this can be refined by considering the whole e-mail content, including header and body. An e-mail is classified as spam when its informative content is similar or close to that of another spam. So we are interested in a distance between 2 e-mails ensuring that when they are close, we can conclude that they have similar informative content without distinction between header and body. One can wonder about the formal meaning of that 'informative content' concept. Kolmogorov (also known as Kolmogorov-Chaitin) complexity is a strong tool for this purpose. In its essence Kolmogorov's work considers the informative content of a string s as the measure of the size of the ultimate compression of s, noted $\mathcal{K}(s)$.

In this paper we aim at showing that Kolmogorov complexity and its associated distance can be pretty reliable in classifying spam. Most important is that this can be achieved:

- without any body analysis
- without any header analysis

The originality of our approach is the use of support vector machines for classification such that we represent every e-mail in the training set and in the testing set as a multi-dimensional real vector. This can be done by considering a set of typical e-mails previously identified as spam or ham.

The rest of this paper is organized as follows: we briefly review the Kolmogorov theory and its very meaning for a given string s. Then we describe the main idea behind practical applications of Kolmogorov theory which is defining a suitable information distance. Since $\mathcal{K}(s)$ is an ideal number, we explain how to estimate $\mathcal{K}$ and the relationship with commonly used compression techniques. We then exhibit our first experiment with a simple k-nearest neighbors algorithm and show why we move to more sophisticated tools. After introducing support vector machine classifiers, we describe our multi-dimensional real vector representation of an e-mail. We exhibit and comment our new results. We discuss related approaches before concluding and outlining future perspectives.

What is Kolmogorov complexity?

The data a computer deals with are digitalized, namely described by finite binary strings. In our context we focus on pieces of text. Kolmogorov complexity $\mathcal{K}(s)$ is a measure of the descriptive complexity contained in an object or string s. A good introduction to Kolmogorov's complexity is contained in (Kolmogorov 1965) with a solid treatment in (Li & Vitányi 1997; Kirchherr, Li, & Vitányi 1997). Kolmogorov's complexity is related to Shannon entropy (Shannon 1948) in that the expected value of $\mathcal{K}(s)$ for a random sequence is approximately the entropy of the source distribution for the process generating this sequence. However, Kolmogorov's complexity differs from entropy in that it is related to the specific string being considered rather than to the source distribution. Kolmogorov complexity can be roughly described as follows, where $\mathcal{T}$ represents a universal computer (Turing machine), p represents a program, and s represents a string:

$\mathcal{K}(s)$ is the size $|p|$ of the smallest program p s.t. $\mathcal{T}(p) = s$

Thus p can be considered as the essence of s since there is no way to get s with a shorter program than p. It is logical to consider p as the most compressed form of s. The size of p, $\mathcal{K}(s)$, is a measure of the amount of information contained in s. Consequently $\mathcal{K}(s)$ is the lower bound limit of all possible compressions of s: it is the ultimate compression size of the string. Random strings have rather high Kolmogorov complexity on the order of their length, as patterns cannot be discerned to reduce the size of a program generating such a string. On the other hand, strings with a large amount of structure have fairly low complexity and can be massively compressed.
Universal computers can be equated through programs of constant length, thus a mapping can be made between universal computers of different types, and the Kolmogorov complexity of a given string on two computers differs by known or determinable constants.
In some sense, this complexity is a kind of universal characteristic attached to a given data. For a given string s, $\mathcal{K}(s)$ is a strange number which cannot be computed just because it is an ideal lower limit related to an ideal machine (Universal Turing machine or calculator). This is a major difficulty which can be overcome by using the fact that the length of any program producing s is an upper bound of $\mathcal{K}(s)$. We develop this fact in a further section. But now we need to understand how to build a suitable distance starting from $\mathcal{K}$. This is our aim in the next section.

Information distance

When considering data mining and knowledge discovery, mathematical distances are powerful tools to classify new data. The theory around Kolmogorov complexity helps to define a distance called 'Information Distance' or 'Bennett's distance' (see (Bennett *et al.* 1998) for a complete study). The informal idea is that given two strings or files, a and b, we can say:

$$\mathcal{K}(a) = \mathcal{K}(a \setminus b) + \mathcal{K}(a \cap b)$$
$$\mathcal{K}(b) = \mathcal{K}(b \setminus a) + \mathcal{K}(a \cap b)$$

It is important to point out that those 2 equations do not belong to the theoretical framework but are approximate formula allowing to understand the final computation. The first equation says that a's complexity (its information content) is the sum of the proper a's information content denoted $a \setminus b$, and the common content with b denoted $a \cap b$. Concatenating a with b yields a new file denoted $a.b$ whose complexity is $\mathcal{K}(a.b)$:

$$\mathcal{K}(a.b) = \mathcal{K}(a \setminus b) + \mathcal{K}(b \setminus a) + \mathcal{K}(a \cap b),$$

since there is no redundancy with Kolmogorov compression. So the following number:

$$m(a, b) = \mathcal{K}(a) + \mathcal{K}(b) - \mathcal{K}(a.b) = \mathcal{K}(a \cap b)$$

is a relevant measure of the common information content to a and b.
Normalizing this number in order to avoid side effects due to strings size helps in defining the information distance:

$$d(a, b) = 1 - m(a, b)/max(\mathcal{K}(a), \mathcal{K}(b))$$

Let us understand the very meaning of d. If $a = b$, then $\mathcal{K}(a) = \mathcal{K}(b)$ and $m(a, b) = \mathcal{K}(a)$ thus $d(a, b) = 0$. On the opposite side, if a and b do not have any common information, $m(a, b) = 0$ and then $d(a, b) = 1$. Formally, d is a metric satisfying $d(a, a) = 0$, $d(a, b) = d(b, a)$ and $d(a, b) \leq d(a, c) + d(c, b)$ over the set of finite strings. d is called the information distance or the Bennett distance. If $d(a, b)$ is very small, it means a and b are very similar. $d(a, b)$ close to 1 means a and b have very few information in common. This is exactly what we need to start with classification. As we understand now, the basic quantity we need to compute or estimate for a given string s is $\mathcal{K}(s)$. This is addressed in the next section.

Kolmogorov complexity estimation

In this section we are interested in adapting the formal tool introduced so far to real-world applications. We have mentioned above that the Kolmogorov complexity exact value can never be computed. Though $\mathcal{K}(s)$ being the lower limit of all possible compressions of s means that every compression $C(s)$ of s approximates $\mathcal{K}(s)$. A decompression algorithm is considered as our universal Turing machine $\mathcal{T}$ such that: $\mathcal{T}(C(s)) = s$.
Hence it is obvious that we focus our attention on lossless

compression algorithms to preserve the validity of the previous equality. Fortunately there exists a variety of such algorithms coming from the 'compression' community. Discussing those tools falls out of the scope of this paper, and we briefly overview some classical compression tools available on the market. On one side there are formal lossless compression algorithms: `LZW` standing for Lempel-Ziv-Welch (Welch 1984), Huffman (Huffman 1952), Burrows-Wheeler (Burrows & Wheeler 1994). On the other side there are real implementations adding some clever manipulations to the previous algorithms:

- Unix `compress` utility based on `LZ` which is a less elaborated version of `LZW`
- `zip` and `gzip` which are a combination of `LZ` and Huffman encoding (for instance, Adobe Reader contains an implementation of `LZW`)
- `bzip2` which first uses Burrows-Wheeler transform then a Huffman coding.

The `bzip2` tool is known to achieve interesting compression ratios and we choose it to approximate Kolmogorov complexity. In the following, instead of dealing with the ideal number $\mathcal{K}(s)$ where s is a file, we deal with the size of the corresponding compressed file. The better the algorithm compresses the data s, the better the estimation of $\mathcal{K}(s)$. When replacing $\mathcal{K}(s)$ by $C(s)$, it becomes clear that the previous information distance is not anymore a true mathematical distance but remains sufficient for our purpose.

Our first tests with k-nn

In order to quickly validate our approach, we start from a classical k-nearest-neighbors (k-nn) algorithm where all neighbors of a given incoming e-mail equally contribute to determining the actual class to which it belongs (either spam or ham). The training set S is constituted of both spam and ham e-mails. Complexity estimation is obtained by `bzip2` compression tool whose C source code is freely available (`http://www.bzip.org/`) and easy to integrate. This technique performs without information loss and is quite good in term of runtime efficiency, which is exactly what we need.
Basically, starting from an incoming e-mail, we compute its k-nearest neighbors belonging to a set of witness e-mails previously classified as spam/ham. Then we choose the most frequent class among those k neighbors as the class for the incoming e-mail. The distance we use is just the information distance estimated through `bzip2` compression. Our algorithm is implemented in C and our programs (source and executable) are freely available on request.
Using the previous software, our experimentation has been run over a set of 200 new e-mails to classify. We then performed 4 tests series using 4 different training sets with cardinalities as below:

- 25 spam/25 ham,
- 50 spam/50 ham,
- 75 spam/75 ham,
- 100 spam/100 ham.

The only parameter we tune is k and we choose k as 11, 21, 31, 41, 51, 61, 71. As expected, when increasing k to a certain threshold, the results quality decreased. That is why finally we choose the value of k as the square root of the training set cardinality. This is just an empirical value which works well. Below we provide only one graphic (with 200 checked e-mails) which gives an intuition of our results:

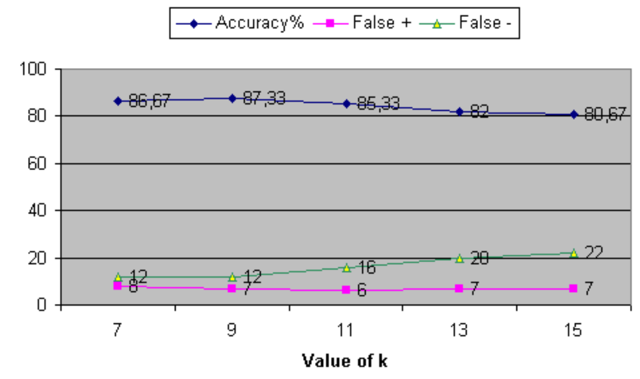

Figure 1: 50 spam/50 ham witness e-mails.

Visual inspection of our curve provides clear evidence that information distance really captures an underlying 'semantics'. Since in our samples there are e-mails in diverse languages (French, English), we can think that there is a kind of common underlying information structure for spam. The accuracy rate remains in the range of 80%-87%, whatever the number of witness e-mails, and the false negative level remains low. Up to now there is no existing filtering method ensuring a perfect 100% accuracy, thus false positive level implies that the individual user still needs to check his junk folder to avoid losing important e-mails.
Since our first results are quite encouraging, we decide to move to a completely different and more sophisticated technique. Instead of investigating the k-nn algorithm tuning, we work with a powerful training component, namely Support Vector Machines (SVM), and we adapt our implementation. We briefly introduce SVM in the next section.

SVM for spam classification

Support vector machines are powerful classifiers. It is out of the scope of this paper to review SVM (see (Burges 1998)) and we only give a flavor. Given a set of k-dimensional vectors x_i, each one labeled 1 or -1 depending on their classification. Given a problem, a discriminating hyperplane w is one that creates a decision function satisfying the following constraint for all i: $y_i(x_i.w + b) - 1 \geq 0$.
The learning problem is called *linearly inseparable* in case of no existing such hyperplane. This drawback can be dealt with following some strategies. Indeed Support Vector Machines can use a nonlinear kernel function defining a new inner-product on the input space. This inner product may be used to calculate many higher-power terms of samples combinations in the input space. This yields a higher-dimensional space such that when it is large enough, there will be a separating hyperplane. An SVM tool is controlled

by 2 parameters. The main one relates to the kernel function. Radial Basis Function (RBF) kernel is widely used since it somewhat captures other kernel functions. It takes the value 1 in case of two equal input vectors. Contrarily, its value slowly decreases towards 0 in a radially symmetric fashion: $K(x_i, x_j) = e^{-||x_i - x_j||^2/2\alpha^2}$.
Here, α is a parameter controlling the decreasing rate and is set priorly to the SVM learning procedure. Other parameters may be considered as well but it is not our aim to tune such parameters in the present paper. That is why we leave this task to the SVM package we use since there is a way to automatically provide suitable values for these parameters.
As we understand, running an SVM algorithm depends on a vector representation of e-mails. We obtain this by exploiting a quite simple idea. We choose a set of typical e-mails: $Typ = \{t_i | i \in [1, n]\}$ mixing 50% spam and 50% ham. For every e-mail m not belonging to Typ we compute the information distance: $m_i = d(m, t_i)$.
We get $\#Typ$[1] coordinates, building a $\#Typ$-dimensional vector representing m. The choice of Typ sets up the dimension of the vector space we deal with. Later on we will see that Typ's quality is crucial for the filter performance.

In terms of implementation, we use the LIBSVM software (Chih-Chung Chang & Chih-Jen Lin 2001). When it comes to evaluate the relevance of a spam filtering system, the standard accuracy rate (which is a usual measure for classification purpose) is not really sufficient since all errors are treated on equal footing. If we have 90% of accuracy rate, it is possible to have 10% of errors only by misclassifying the ham. It means that your inbox is clean but you have to check your junk box to get the missing ham. So it is important for an end-user to know the rate of ham (or legitimate e-mails) lost during the process. If this rate (False Positive Rate) is very low (for instance around 1%), then there is no need to deeply examine the junk box. Vice-versa if the number of not identified spam is important, then a lot of spam remain in the inbox: so we want a high rate of identified spam (True Positive Rate). Another interesting value is the number of spam properly identified among the total number of e-mails considered as spam (true spam and false positives). This value is the Positive Predictive Value (or precision). Let us give formal definitions:

- fn = number of spam identified as ham
- fp number of ham identified as spam
- s number of spam in the test set
- h number of ham in the test set ($s + h$ is the test set cardinality)
- accuracy $= 1 - (fn + fp)/(s + h)$
- FPR = fp/h (False Positive Rate or fall out)
- TPR = $(s - fn)/s$ (True Positive Rate or recall)
- PPV = $(s - fn)/(s - fn + fp)$ (Positive Predictive Value also known as precision).

[1]Here, $\#Typ$ denotes the cardinality of the set Typ.

It is quite clear that those numbers are not independent and sometimes, it can be interesting to represent TPR as a function of FPR (roc analysis). Due to space limitations, we do not present graphical representation of our results.

Our results with SVM

We have run a set of experiments on the publicly available Spamassassin corpus. This set is constituted of 9348 e-mails of which 2397 are spam. We performed no pre-processing on those e-mails that we consider as raw input data for our system. Below is our protocol:

- First of all, we have defined 4 *training sets* of 50, 100, 150 and 200 e-mails.
- We have then defined 5 *sets of typical e-mails* with cardinality 8, 10, 12, 16 and 20, each one on a 50/50% spam/ham basis.
- Those 5 typical sets gave rise to 5 experiment types with the same training and testing sets: each type is associated to an 8, 10, 12, 16 and 20 dimensional vector space.
- Then instead of working on the full dataset, we have chosen 3 *test sets*: 2 with 500 and one with 1000 e-mails of both types. We give here the results for the final 1000 set.

Our experiments series are aimed to:

- allow to choose the most effective vector space dimension
- allow to choose the most effective training set cardinality.
- validate the combination of Kolmogorov complexity and SVM as a really effective framework for spam filtering.

Table 1 presents our results (all numbers are percentages, the testing set contains 1000 new e-mails to classify).

train50	**8**	**10**	**12**	**16**	**20**
Accuracy	94.60	95.60	95.40	94.60	95.00
FPR	1.60	2.80	4.00	5.60	5.20
TPR	90.80	94.00	94.80	94.80	95.20
train100	**8**	**10**	**12**	**16**	**20**
Accuracy	94.60	96.20	95.40	94.80	94.80
FPR	1.60	2.40	2.40	1.60	1.20
TPR	90.80	94.80	93.20	91.20	90.80
train150	**8**	**10**	**12**	**16**	**20**
Accuracy	94.80	94.60	94.60	96.40	96.20
FPR	1.20	2.40	2.40	2.40	2.40
TPR	90.80	91.60	91.60	95.20	94.80
train200	**8**	**10**	**12**	**16**	**20**
Accuracy	93.80	93.40	93.60	93.80	93.40
FPR	1.60	1.60	2.00	1.60	1.60
TPR	89.20	88.40	89.20	89.20	88.40

Table 1: Classification results with `bzip2`

In our experiment, and except with a training set of 150 e-mails, it is clear that the best dimensional representation

should be in the range of 8-10. We think there is a relationship between the size of the vector representation and the training set size: It is well known that over-fitting can occur with a huge training set, thus reducing the prediction power of the classifier. In fact when we increase the vector dimension (e.g. 16 or 20 in our experiment), we increase the quantity of information carried by an e-mail. It is realistic to consider that relatively small training sets perform better than bigger ones since they are less sensitive to over-fitting. This relationship needs to be more deeply investigated.
Regarding the training set size, 100 seems to be quite effective. It can be seen from our results that when dealing with a bigger training set (150 or 200), some kind of over-fitting effect harms the classifier accuracy and in particular the harms the false positive and true positive rates. Obviously a smaller training set (50 e-mails) is not sufficient to output a precise classifier via the SVM tool. Moreover our experiment shows that, in that case, the false positive rate reaches the worst values as the vector space dimension increases (12, 16 and 20). It is worth noting that, except for the smaller training set, our accuracy rates range from 93% to 97% with FPR between 1% and 3%, which is usually quite acceptable.

Despite the fact that more investigation has to be performed, it is obvious that the pair Kolmogorov complexity-SVM provides a clean theoretical framework to accurately deal with practical applications like spam filtering. In that case, our simple implementation is equivalent in terms of performance to more mature tools (such as SpamBayes or Spamassassin). It becomes definitely clear that a deep analysis of the incoming e-mail itself or a clever pre-processing are not a prerequisite to get accurate results. Better performances could be expected (e.g. by combining diverse approaches), however we are not sure this is feasible since Kolmogorov complexity theory is a strong substratum. Of course there is room for improving our model due to the number of parameters controlling it!

Related works

Despite their relative novelty, complexity-based approaches have been proved quite successful in numerous fields of IT: data mining, classification, clustering, fraud detection, and so on. Within the IT security field, we can refer to works of (Kulkarni & Bush 2001; Bush 2002; 2003), and more recently (Wehner 2006): they are mainly devoted to analyzing data flow coming from network activities and to detecting intrusion or virus. From a spam filtering perspective, using compression paradigm is not a completely new idea, despite the fact that our way to mix Kolmogorov complexity with an SVM training component is quite new (as far as we know and except other applications such as protein sequence classification (Kocsor *et al.* 2006)).
Probably the most closely related work is that of Spracklin and Saxton (Spracklin & Saxton 2007): they use Kolmogorov complexity without referring to the information distance. In each e-mail, they just consider the body which is first pre-processed and cleaned up (only lower case letters, removal of common words and HTML tags, etc.). Then each word is converted into 0 if it appears frequently in spam or 1 if it appears more frequently in ham. The final result is a string of 0 and 1 which is then compressed. If the complexity is below a certain threshold the e-mail is classified as ham, otherwise as spam. Their accuracy rates vary in the range of 80%-96% depending of their test sets. This is a good result similar to ours, with the difference that we do not pre-process our e-mails. In term of complexity, our process is much simpler since we only compute distances.
Work of Bratko et al. (Bratko *et al.* 2006) also has to be cited. It is not exactly based on Kolmogorov complexity but is compression-based. They use an adaptive statistical data compression process. The main idea is to consider the messages as issued by an unknown information source and to estimate the probability of a symbol x to be produced. Such a probability then exhibits a way to encode the given symbol and to compress. So in order to filter spam, it is sufficient to compress the training ham set into a file Ham, and to do the same for the training spam set getting a file $Spam$. When a new e-mail m arrives, it is added to the ham folder and the resulting folder is compressed into a file $Ham + m$. The same is done for the spam folder getting $Spam + m$. If the difference between $Spam + m$ size and $Spam$ size is small, it means m does not bring new information to the training spam and is likely to be a spam. Otherwise m is considered as a ham. This method, which does not use the information distance, provides good results on the standard databases whatever the compression model. Our rates are currently similar but we have not yet investigated the whole collection of available databases.

Conclusion and Future works

In this paper we have investigated the anti-spamming field and have considered the Kolmorogov complexity of an e-mail as a tool for classifying it as spam or ham. In order to quickly validate our idea, we first implemented a basic version of k-nearest neighbors algorithm, using the information distance deduced from $\mathcal{K}$ as a closeness measure. Our tests have been processed on relatively small data sets. Still, the first obtained results clearly show that the compression distance is meaningful in that situation. This is not completely coming as a surprise: the seminal works of (Bennett *et al.* 1998) then (Cilibrasi & Vitányi 2005) already paved the way and highlighted the real practical power of compression methods. One major advantage in using compression for e-mail classification is that there is no need to deeply analyze the diverse parts of an e-mail (header and body). Our algorithm can of course be refined, and simply having a more powerful compression technique would probably bring more accurate results. Multiple complexity estimators could also be combined in order to improve discrimination accuracy. A more sophisticated machine learning component based on SVM could then be implemented to further investigate our compression-based classification method.
In the present work using SVM required a real vector representation of e-mails and it appears that 10 is a suitable dimension for the vector space model. Then we trained our SVM on diverse e-mail sets: by examining our experimental results it appears that a suitable size for the training set is in the range of 100. On our test sets (coming from Spa-

massassin repository), we got accuracy rates in the range of 93%-97%, with false positive rates between 1% and 2%. It is quite clear that the compression method coupled with the SVM training component is successful, and our results emphasize the idea that there is no need to separate the header from the body and to analyze them separately. A great advantage of this kind of 'blind' techniques is that there is:

- no need to update a dictionary or a blacklist of domain names. The system automatically updates when the training base evolves over time (i.e. the database of junk/legitimate e-mails which is basically unique to a given user): it can be trained on a per-user basis, like Bayesian spam filtering.
- no need to pre-process the e-mails (like tokenization, HTML tag and capital letters removal).

A similarity with Bayesian filtering is that complexity based methods do not bring any explanations about their results. Fortunately this is generally not a requirement for an e-mail filtering system. In that particular case it cannot be considered as a drawback. Even if some commercial softwares claim 99% of success in spam elimination (e.g. see `http://www.liveprism.com/`), it is well known that the end users are not entirely satisfied by the current performances of anti-spam tools. For instance in (SPAMfighter 2007) people estimate around 15% the amount of daily spam they receive after the spam filters have done their job. It means 85% of accuracy which is clearly outperformed in our experiments. It is quite clear that compression based approaches are not the ultimate Graal. Though we strongly believe that when mixed with other classical strategies they will definitely bring a step further in the spam filtering field. For us it remains to investigate how to overcome the fact that we only estimate the Kolmogorov complexity. Using compression makes information distance calculation approximative since $d(m, m)$ is definitely not 0 but a small value between 0.1 and 0.3. This error has to be properly managed to provide accurate vectors for the training components. This is our next step to conceive a more accurate system and to ultimately tend towards a 'spam-free' internet!

Acknowledgment

The authors are grateful to Ivan José Varzinczak for his help in integrating the `bzip2` code to the used programs.

References

Bennett, C.; Gacs, P.; Li, M.; Vitányi, P.; and Zurek, W. 1998. Information distance. *IEEE Transaction on Information Theory* 44(4):1407–1423.

Bratko, A.; Cormack, G. V.; Filipic, B.; Lynam, T. R.; and Zupan, B. 2006. Spam filtering using statistical data compression models. *Journal of Machine Learning Research* 7:2673–2698.

Burges, C. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* 2(2):121–167.

Burrows, M., and Wheeler, D. 1994. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation.

Bush, S. F. 2002. Active virtual network management prediction: Complexity as a framework for prediction, optimization, and assurance. In *Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE)*, 534–553.

Bush, S. F. 2003. Extended abstract: Complexity and vulnerability analysis. In *Complexity and Inference*.

Chih-Chung Chang, and Chih-Jen Lin. 2001. Libsvm : a library for support vector machines. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cilibrasi, R., and Vitányi, P. 2005. Clustering by compression. *IEEE Transaction on Information Theory* 51(4).

Graham, P. 2002. A plan for spam. Available online at `http://www.paulgraham.com/spam.html`.

Huffman, D. 1952. A method for the construction of minimum reduncancy codes. In *Proceedings of the IRE*.

Kirchherr, W.; Li, M.; and Vitányi, P. 1997. The miraculous universal distribution. *MATHINT: The Mathematical Intelligencer* 19(4).

Kocsor, A.; Kertész-Farkas, A.; Kaján, L.; and Pongor, S. 2006. Application of compression-based distance measures to protein sequence classification: a methodological study. *Bioinformatics* 22(4):407–412.

Kolmogorov, A. N. 1965. Three approaches to the quantitative definition of information. *Problems in Information Transmission* 1(1):1–7.

Kulkarni, P., and Bush, S. F. 2001. Active network management and kolmogorov complexity. In *OpenArch 2001*.

Li, M., and Vitányi, P. 1997. *Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag.

Lowd, D., and Meek, C. 2005. Anti-spam products give unsatisfactory performance. In *Proceedings of the Second Conference on E-mail and Anti-spam (CEAS)*, 125–132.

Pearl, J., and Russell, S. 2003. Bayesian networks. In A., A. M., ed., *Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press. 157–160.

Shannon, C. 1948. A mathematical theory of communication. *Bell System Technical Journal* 27:379–423, 623–656.

SPAMfighter. 2007. Anti-spam products give unsatisfactory performance. Available online at `http://www.spamfighter.com/News_List_Other.asp?D=2007.7.31`.

Spracklin, L., and Saxton, L. 2007. Filtering spam using kolmogorov complexity estimates. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW)*, 321–328.

Wehner, S. 2006. Analyzing worms and network traffic using compression. Available online at `http://arxiv.org/abs/cs.CV/0504045`.

Welch, T. 1984. A technique for high performance data compression. *IEEE Computer* 17(6).