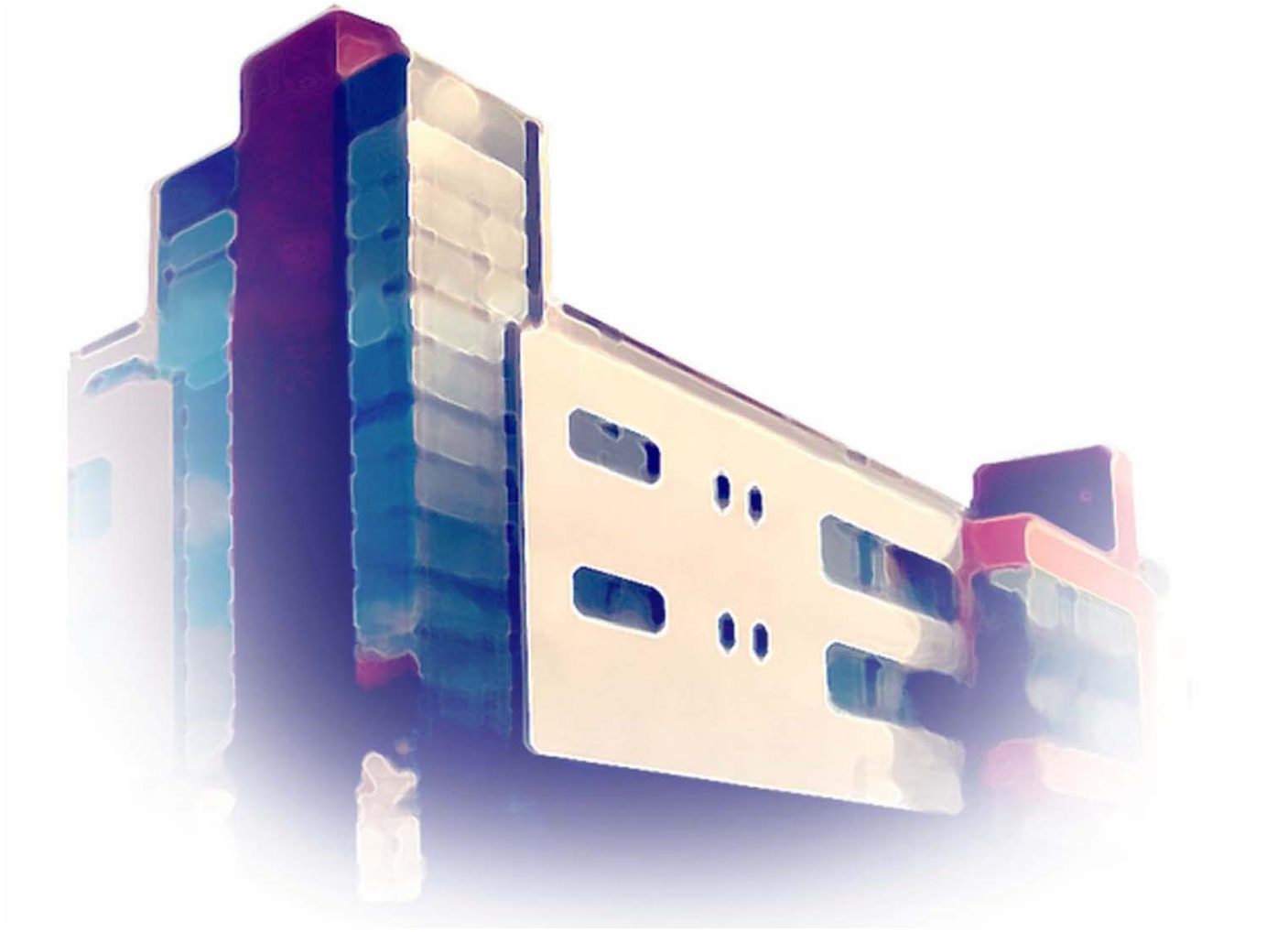
Christian Kaul, Johannes Knabe, Tobias Lang, Volker Sperschneider

Filtering Spam E-mail with Support Vector Machines



PICS

Publications of the Institute of Cognitive Science

Volume 8-2004

ISSN: 1610-5389

Series title: PICS
Publications of the Institute of Cognitive Science

Volume: 8-2004

Place of publication: Osnabrück, Germany

Date: September 2004

Editors: Kai-Uwe Kühnberger
Peter König
Petra Ludewig

Cover design: Thorsten Hinrichs

Filtering Spam E-mail with Support Vector Machines

Christian Kaul, Johannes Knabe, Tobias Lang and Volker Sperschneider

Institute of Cognitive Science, University of Osnabrück

chrisskaul@web.de, jknabe@uni-osnabrueck.de,
tlang@uni-osnabrueck.de, sper@informatik.uni-osnabrueck.de

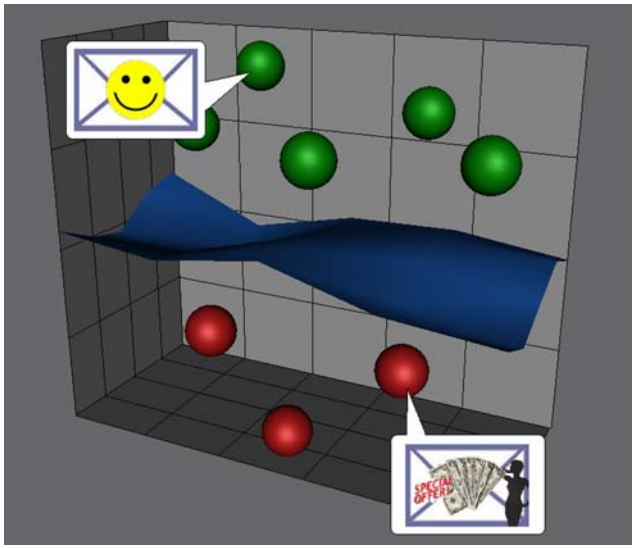
Abstract

It is investigated in how far support vector machines are suitable for filtering spam e-mail. Support vector machines have several advantages in comparison to other machine learning approaches and have been proven to be an efficient tool for classification of linguistically preprocessed text. A software program has been developed that learns the individual classification criteria of its user and filters e-mail accordingly. The complete process is fully autonomous, based solely on sample e-mails. In an evaluation the program is compared to conventional approaches showing advantages in reliability and maintenance.

Introduction

The problem of text categorization arises in many different areas such as preprocessing news agency stories and data mining. Nowadays, a prominent task is to classify incoming e-mails. An e-mail user wants his or her e-mail to be sorted to different folders according to their contents, filtering out undesired spam e-mail.

Text categorization is one of the major application areas of support vector machines (SVM). In an often cited experiment, Joachims (1998) achieved better results with an SVM algorithm than with standard methods in the classification of labelled Reuter's news stories that were to sort into categories such as money market, earning etc.



An SVM constructs a hyperplane in the feature space and thereby separates spam from desired e-mail.

In our study, we investigate the rapidly growing problem of spam e-mail. A classifier makes the binary decision whether an incoming e-mail is handled as

spam or as desired e-mail. Though many different approaches to this problem exist, most of them are using explicit rules that need to be specified by the program or by the user (there are, however, also approaches that use self-learning probabilistic methods such as Bayesian Nets). Our goal was to develop a powerful spam-filter that learns fully autonomously from individual examples provided by the user, i.e. it learns to imitate its user's classification strategy. Techniques from neuroinformatics such as neuronal networks fulfill these criteria. In particular, SVMs are well suited for this task, as they have outstandingly good generalization performances and are based on sound mathematical grounds which makes it straightforward to use them.

In the following, we will give a short introduction into the theory of SVMs for the non-familiar reader. Section two will discuss different possibilities how to preprocess text data to be used by support vector techniques. In a third section, the architecture of our program is described as it was presented at CeBIT 2004. Section four discusses and evaluates its advantages and disadvantages. Finally, we assess the performance of our program and give an outlook of how it could be further improved.

Theory of Support Vector Machines[1]

Binary classification is one of the most important tasks that can be treated using neural networks. Input vectors are to be classified into two classes. This is exactly the situation when we want to separate e-mails into spam and non-spam.

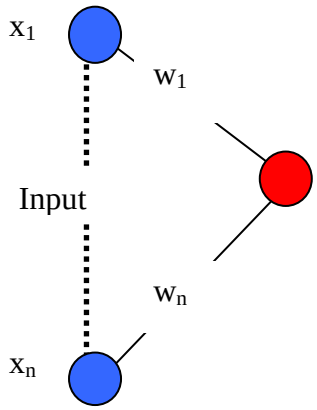
Neural networks use a finite training set consisting of vectors $x \in \Re^n$ of dimension n together with corresponding class labels $d \in \{+1, -1\}$. Vectors with class label +1 are called positive instances; vectors with label –1 are called negative instances. A neural network is trained to correctly classify such training vectors. We expect appropriate generalization to take place after training in the sense that the neural network also correctly classifies (most) unseen instances. Aspects that influence training success and generalization ability are: *(i)* The network must be powerful enough to allow for a (approximate) solution of the task under consideration. This can be achieved by choosing a model with many parameters (neurons, hidden layers, weights). *(ii)* The more parameters we have the more expensive is the process of training. An efficient training algorithm is required to achieve rapid training, particularly in situations where a network is to be retrained very often (as is the case with the Spam problem). *(iii)* The network must be not too powerful in order to prevent overfitting.

Model selection is a severe problem. In the context of feed-forward networks, for example, pruning algorithms reduce the amount of neurons and weights, feature selection appropriately designs the input interface of the network, genetic algorithms optimize network topology. Nevertheless, finding an appropriate feed-forward network for a given learning task may be a difficult and time-consuming task.

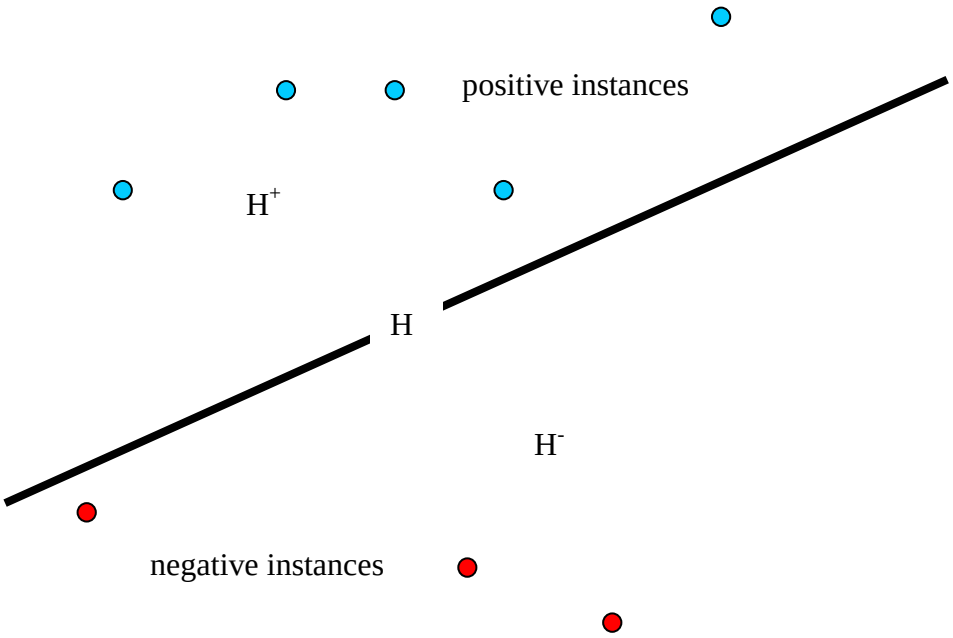
Support vector machines solve all of these problems in an elegant and mathematically well-founded manner. From an architectural point of view, a SVM looks like a simple Perceptron, a device consisting solely of n input neurons that receive the coordinates of an input vector x, and a single output neuron that predicts the class label of x by summing up the coordinates of x in a weighted manner and firing if this weighted sum exceeds a certain

[1] The reader familiar with SVMs may skip this section.

threshold. Having real-valued weights w_1, …, w_n and threshold –b, the model thus looks as follows:



Geometrically expressed, class separation by a Perceptron means that the hyperplane H = { x ∈ $\Re^n$ / $\langle w,x\rangle + b = 0$ } separates positive from negative instances: Positive instances fall into the positive halfspace H^+ = { x ∈ $\Re^n$ / $\langle w,x\rangle + b > 0$ }, and negative instances fall into the negative halfspace H^- = { x ∈ $\Re^n$ / $\langle w,x\rangle + b < 0$ } defined by H. Here, $\langle w,x\rangle$ denotes the usual inner product in n-dimensional real space, that is $\langle w,x\rangle = \sum_{i=1}^{n} w_i \cdot x_i$.

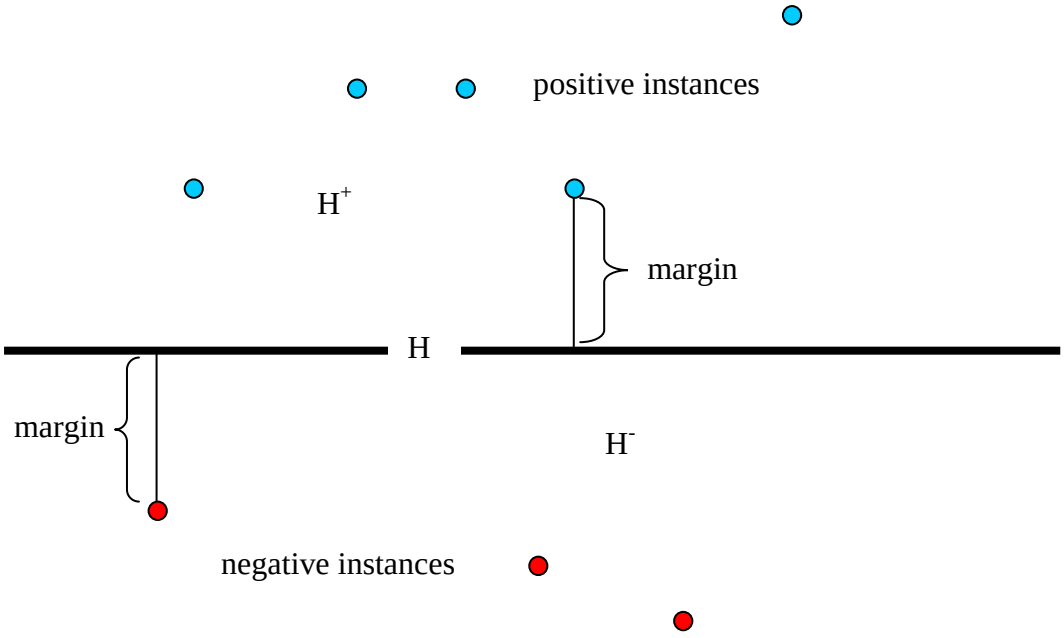


Training of a Perceptron usually is done by the Perceptron algorithm that changes the actual weight vector w into w + x, when seeing a randomly chosen positive instance x that is incorrectly classified into H^-, and into w – x, when seeing a randomly chosen negative instance that is incorrectly classified into H^+. The famous Perceptron Convergence Theorem states that for a linearly separable training set finally a weight vector is found that indeed separates positive from negative instances.

Problems with Perceptrons are: Training sets often are not linearly separable. Embedding them into a larger input space (by defining further features) often makes them linearly separable,

but leads to more weights thus leading to worse generalization and longer training times. Convergence time of the Perceptron algorithms may be long.

In Learnability Theory, generalization ability of models can be quantitatively defined. In the case of the Perceptron it can be shown that generalization depends on the so-called margin of the hyper-plane H with respect to the training set, that is, on the minimum distance points from the training set have to H, but it does not depend on the input dimension n. The example below supports the idea that margin maximization improves generalization. The indicated hyperplane has a greater margin than the formerly proposed one – and seems to separate positive from negative examples in a more robust manner:



SVMs compute the separating hyperplane H having maximal margin with respect to the training set by solving (in an efficient manner) a corresponding standard quadratic optimization problem that results from a Lagrangian formulation of the problem and transformation into its dual problem (technical details are omitted).

Generalization ability is guaranteed by theorems from statistical learnability theory like the following:

> Let labelled data (x,d) be drawn according to a fixed, by unknown distribution P(x,d) with the constraint that only vectors $x \in \Re^n$ of norm less than a fixed value R are drawn. Define for each hyperplane H the generalization error $Err_P(H)$ with respect to the distribution H as the probability to draw a labelled vector (x,d) which is misclassified by hyperplane H. Correspondingly, define for each training set T consisting of m labelled vectors (x,d) the empirical error $Err_{emp}(T,H)$ as the number of elements of T that are misclassified by H, divided by m. Finally, the probability of occurrence of T, P(T), is defined assuming that labelled vectors (x,d) are independently drawn according to distribution P. Note that $Err_{emp}(T,H)$ can be simply calculated, given training set and hyperplane H, whereas $Err_P(H)$ cannot be

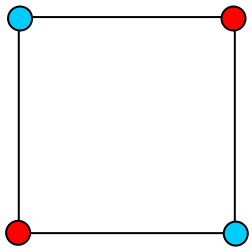
calculated, unless the distribution P is known (and simple enough). Nevertheless, $Err_P(H)$ is the number we are interested in.

Now the following holds true: Given a number δ between 0 and 1 (called confidence), with probability greater that 1 - δ any training set T of m labelled vectors drawn independently according to P has the property that any hyperplane H with margin at least μ with respect to T has generalization error at most

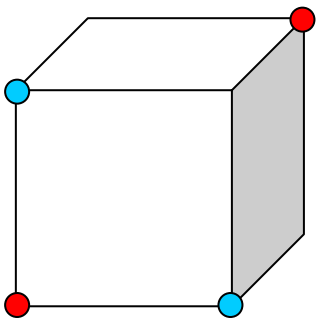
$$\frac{2}{m} \cdot \left(\frac{64 \cdot R^2}{\mu^2} \cdot \log\frac{e \cdot m \cdot \mu}{8 \cdot R^2} \cdot \log\frac{32 \cdot m}{\mu^2} + \log\frac{4}{\delta}\right)$$

Ignoring mathematical details of the approach and the formula, the essential message is that quality of generalization scales with R^2/μ^2 and 1/m (m = Number of training examples). The additional term log(4/δ) simply expresses that the more confident we want to be the more liberal we must be with the expected generalization error. So, the greater the margin is, the better generalization is. The appearance of radius R in R^2/μ^2 is easily explained: Scaling up vectors by a constant factor c would increase margin also by a factor of c, but obviously would not affect generalization error. So, the quotient R/μ can be the only meaningful measure for generalization quality.

A second message of the formula is that input dimension n has no influence on generalization ability. This means that we must not worry too much about the amount of input features. In particular, having a not linearly separable task, it offers the opportunity to embed input space into an enlarged feature space without loss in generalization ability thus increasing the chance to obtain a linearly separable task. As a simple example, consider the non-separable XOR-problem in 2-dimensional space. Positive instances (0,1) and (1,0) cannot be linearly separated from negative instances (0,0) and (1,1).



Now add the additional feature that is simple the product (logical AND) of the first two coordinates to obtain positive instances (0,1,0) and (1,0,0), and negative instances (0,0,0) and (1,1,1). The problem becomes linearly separable.



Embedding input space $\Re^n$ by a function Φ into some feature space $\Re^N$ with N that usually is much greater that n (even Hilbert spaces H of infinite dimension are allowed as feature spaces) improves separability without causing problems with respect to generalization ability. Nevertheless, another problem comes up: Finding hyperplanes with maximal margin involves an extensive amount of computations of inner products. Obviously, it makes a difference whether this has to be done in input space of low dimension n or in feature space of high dimension N. One final trick also solves this problem: Embeddings $\Phi: \Re^n \rightarrow \Re^N$ that are used in practise usually have the property that the inner product $\langle\Phi(x),\Phi(y)\rangle$ between two feature vectors can be calculated from the inner product $\langle x,y\rangle$ between the input vectors and then applying a simply calculable kernel function K: $\langle\Phi(x),\Phi(y)\rangle = K(\langle x,y\rangle)$.

Consider an embedding that generalizes the one used above by calculating all products of 0, 1 or 2 input coordinates, that is, all monomials up to degree 2:

$$\Phi(x_1, x_2) = (1, x_1, x_2, x_1x_1, x_1x_2, x_2x_1, x_2x_2).$$

We calculate:

$$\begin{aligned}
&\langle\Phi(x_1, x_2),\Phi(y_1, y_2)\rangle \\
&\quad = \langle(1, x_1, x_2, x_1x_1, x_1x_2, x_2x_1, x_2x_2), (1, y_1, y_2, y_1y_1, y_1y_2, y_2y_1, y_2y_2)\rangle \\
&\quad = 1 + x_1y_1 + x_2y_2 + x_1x_1y_1y_1 + 2x_1x_2y_1y_2 + x_2x_2y_2y_2 \\
&\quad = (1 + x_1y_1 + x_2y_2)^2 \\
&\quad = (1 + \langle(x_1, x_2), (y_1, y_2)\rangle)^2 \\
&\quad = K(\langle(x_1, x_2), (y_1, y_2)\rangle)
\end{aligned}$$

The used Kernel function $K(p) = (1 + p)^2$ is an instance of so-called polynomial kernels. Other kernels are the Gaussian Kernels

$$K_\sigma(x,y) = e^{-\frac{\|x-y\|^2}{2\cdot\sigma^2}}$$

leading to the well-known Radial-Basis-Function networks, or tangens hyperbolicus

$$K_{\Theta,T}(x,y) = \tanh\left(\frac{x^T y - \Theta}{T}\right)$$

leading to the standard feedforward networks (under suitable choice of the parameters Θ and T).

The choice of an appropriate kernel function is essential. Here is the place where specific domain knowledge about the problem under consideration may be integrated into the architecture of a SVM. There are rather elaborate kernel functions that are well suited for the problem of text categorization – though our experience with the Spam problem was that the simpler standard kernels (as e.g. polynomial kernels) are already sufficient to successfully attack the problem.

Summarizing, SVMs are attractive since *(i)* they are grounded on a clear mathematical foundation that explains and guarantees generalization ability *(ii)* rapid training is possible based on standard techniques of quadratic optimization and the choice of suitable kernel functions *(iii)* any feature that appears to be relevant to the problem can be freely used without compromising generalization ability. This made the model attractive for the Spam problem especially due to the fact that a rather simple linguistic preprocessing could be chosen without running into danger of overfitting.

Linguistic Preprocessing

When using SVMs, one basic consideration is the transformation from text documents to mathematical representations. Most prominent is the bag-of-words approach, also called vector space model. In this approach, documents are transformed to vectors whose entries represent the presence of key words in the document. This requires that beforehand a dictionary of key words is chosen from which it is possible to categorize a document. The choice of this dictionary is an essential task since the whole classification will depend upon it.

A document vector, i.e. the mathematical vector representing a text document, may contain different kinds of information. At first, one has to decide what kind of terms are to be represented. It is straightforward to use fully inflected words as terms, so that for example “run” and “running” are handled as two different entries. The more sophisticated choice is to use word stems as representation units so that closely related words are treated as the same. This requires a preprocessing with a stemming algorithm.

Vectors may contain Boolean variables that indicate the presence of the corresponding term in the document. It is also possible to represent the number of occurrences of the corresponding term in the text. Weighting factors may be added that introduce the information of the ratio between the total number of documents and the number of documents containing the specific term so that special attention is given to rare words that are believed to be significant. Finally, document vectors may be normalized to exclude the influence of document length.

Depending on dictionary size, resulting vectors may be very large. This might lead to overfitting when using other techniques, however it is a key characteristic of SVMs to achieve better results proportionally to the amount of given training data. Furthermore, single document vectors are sparse since they do not contain most of the dictionary words. Therefore, in general it should be easy to build a separating hyperplane in the corresponding feature space. Likewise, there are fast techniques for calculating the inner products of sparse vectors. These inner products form the central algorithmic part of a SVM since they compute the similarity of two elements, i.e. of two documents in the context of text categorization. Underlying this approach is the assumption that two documents are similar if they contain the same characteristic words, even if they occur in a different order. The size of the resulting vectors depends upon the size of the dictionary. The choice of an appropriate dictionary is the essential preparation step for text classification by mathematical means. It implicitly contains the domain knowledge by specifying the characteristic terms that make up the classification decision. A possible way to construct a dictionary is to ask experts that name all those terms that are important according to their experience. This corresponds to giving explicit rules. Using such explicit rules is severely criticized since it is error-prone and does not correspond to the goal of creating autonomous systems. Systems should learn themselves what characteristics are needed. In the field of text classification, they shall only be given labeled training examples from which they extract significant terms. This is a form of supervised machine learning. An easy way to do so is to parse all words contained in the training examples and to construct a statistics, measuring the numbers of term occurrences in each category. By comparing the statistics of a given term for the different categories, one may calculate how meaningful this term is for a certain category in comparison to

other categories. On these grounds, one can construct a dictionary that contains only the most significant terms. It is reasonable to assume that uninformative words such as “and”, so called stop-words, are automatically ignored since they should be uniformly distributed over all training examples and categories.

There are various more sophisticated techniques than the ones described here which may lead to better classification results. For example, information about word order is lost in a bag-of-word approach. Much work is currently spent on string kernels that are able to deal with discrete input such as sets of structures (e.g. strings) as opposed to vector spaces as input. For example, they are able to deal with subsequences of words and therefore deliver more sophisticated estimates of the similarity of documents. However, they are not as straight-forward and computationally efficient as the techniques described above. On the other hand, they can renounce on preprocessing steps such as computing word stems since they look directly at subsequences of words.

Architecture of our Program *SpamStop*

In our implementation, we assume that the user provides two folders containing e-mails classified as desired e-mails and spam e-mails respectively. These two sets are then parsed into strings, where one string is a sequence of more than three letters separated from the next by white space. Shorter sequences, punctuation marks and special characters are simply considered white space. The resulting strings are then converted to uppercase and put into two lists, a string not yet present in the corresponding list (desired/spam) is added with an initial counter value of one, and otherwise the counter of the string is increased by one. Currently neither a semantic nor a syntactic mapping algorithm, like e.g. stemming, is used. The simplicity of this approach is one of its major strengths, since we did not want to put our intuitive ideas of tricks which spammers use to bypass simple word matchers, for example “M.E.D.I.C.I.N.E” instead of “medicine”, into the program in order to keep it free of hardwired world knowledge which might – at best – get outdated or – even worse – be misleading. The same holds for semantic mappings, since meaning of a word might depend on its grammatical form and its context. An additional learning algorithm or a steadily updated online database could help here in the future.

Having thus generated lists of most prominent good and bad strings, the lists are then checked for overlaps. In case there is an identical string found in both lists the higher counter gets decreased by the value of the lower counter and the latter one is removed from its list. Now from both lists the 50 strings with the highest counters are selected as future features.

These features are now used to transfer the mails of the two training sets into their numerical representation: a 100-dimensional vector of binary values, each position representing exactly one feature. Every e-mail is parsed as above and for every position of the vector it is checked whether the corresponding feature occurs at least once in it. If so, the position’s value is set to one, otherwise it is set to zero.

Now the core training can start: Different pre-defined Kernel functions are tried with k-fold cross validation, meaning that the training set is divided into k sets and k-1 of them are used for training while the remaining one is used to test the resulting model’s accuracy in classifying the e-mails. The model yielding the highest accuracy is used for classification.

Today there are lots of different e-mail clients competing on the market and running on different operating systems. We wanted our program to run on the user's computer, instead of the server, to reflect the individual classification criteria of users. So we had to be as platform and client independent as possible. Having made these considerations, we decided to implement *SpamStop* in Java for which a runtime environment (*virtual machine*) exists for almost all operating systems. Secondly, *SpamStop* works as a proxy server, i.e. the client does not, as usual, ask the provider's mail server for new e-mails but our program which in turn asks the server. Technically, *SpamStop* listens to a pre-defined port on the computer where it is installed and the client is configured to direct requests to this port on that computer. Thus, it would be easily possible to run it on a remote server as well, provided that either the individually trained model is present on the server or a common model is used for all users. The latter means of course losing the advantage of individual classification criteria but might be applicable in environments where those criteria are homogenous, e.g. in companies. Currently only the widely used Post Office Protocol, version 3, (*POP3*) is supported. The richer but rarely used Internet Message Access Protocol (*IMAP*) is not implemented yet.

In case that there are new e-mails to be received, they are transferred from the provider's server to *SpamStop*, classified as described, and then handed over to the users e-mail client. Every incoming message is converted to its vector representation using the features extracted in the first part of the training as described above and then fed into the SVM using the individual model which resulted from the core part of the training. The SVM returns not only whether the e-mail was on the good or on the bad side of the separating hyperplane in the 100-dimensional "e-mail space", i.e. whether it was more similar to the positive or negative training instances. It also indicates the distance to that hyperplane. Should the e-mail have been on the bad side, i.e. supposedly is spam, this value is used as certainty measure of the classification. It is mapped on a procentual probability, where "far away from the hyperplane" means 100 percent – this is spam for sure – while a small value would yield a lower percentage indicating that the user should have a look at it. At the moment only a warning and the probability are added to the subject line of the corresponding e-mail before it is handed over to the e-mail client. Combined with simple rules of the kind "IF Subject CONTAINS '100% Spam' MOVE_TO Trash", which most e-mail clients support, this is already quite convenient.

Discussion and Program Evaluation

In comparison to other approaches to the classification of e-mail, our program *SpamStop* has several advantages, making it well suited to the user's desires.

Unlike in most other Spam categorization solutions, classification criteria are learned from the user's individual training examples. When these criteria are either handcrafted by experts or learned from e-mails which a (maybe hair-breadth) majority considers spam there is not much of the user's personal attitude left. This becomes crucial in the context of annoyances like stalking where the user receives personalized mail of similar content from varying addresses. Furthermore, one has to account for the possibility that a user wants to receive messages that the majority rejects, such as loan offers.

This example leads to a second advantage of *SpamStop*, its flexibility in the adaptation to new kinds of spam e-mail. Since only training examples influence the classification model, new

kinds of spam e-mail are likely not to be detected by the system. This appears to be a handicap for *SpamStop*, but is in fact one of its advantages. While other, rule-based, systems have to costly develop new rules to account for the new spam types, *SpamStop* needs only to be retrained on an extended example set which contains besides the old examples instances of the new spam type. *SpamStop* will automatically develop new filter criteria adapted to the new situation.

SpamStop is not only based on spam e-mail a user gets, but also makes use of desired e-mail considering them as positive instances. The characteristics of desired e-mail give thereby a second indication system which enhances the classification abilities of the system for difficult and unclear e-mail since it allows for a third classification category: while there were only "guaranteed spam" and "unclear, resp. unlikely to be spam" before, it is now possible to assess an e-mail as "guaranteed desired". Therefore, classification becomes more precise and unambiguously. For the remaining unclear cases, estimation is used as described above.

Although briefly mentioned before, we shall state another advantage of *SpamStop*: its total independence of handcrafted rules. This feature gets extremely important considering the development of spam e-mails. Spammers have long started to adapt to common rules and to undergo them. Therefore rule based systems will always have the problem of an outdated rule basis. To perform the necessary update is both costly and inconvenient. As described, our program presents a better approach to this problem as the updating process is limited to the pure statement of what is a new Spam. In fact, only a constant permutation of every spam e-mail could undergo this. But even then, probably not every word of every new e-mail could be changed while persevering the sense, thus leaving the chance for *SpamStop* to focus on and learn the context surrounding the key word in all variations (e.g. sex, S_E_X, intercourse,…). Other approaches focus on terms either in a boolean way (IF *term* IN_EMAIL THEN *Spam*) or add up points (IF *term* IN_MAIL THEN *spam_value*+=3) and consider it Spam if the sum exceeds a threshold. Since the SVM looks at the position of the vector representing the whole e-mail in relation to the hyperplane, co-occurences of words do matter. That is, the combination of "money", "interest" and "rate" might lead to a 100 percent Spam classification while one of the words alone could be neutral or even a positive indicator.

Performance and Outlook

Although no official benchmark was conducted to assess the performance of our program *SpamStop*, we are able to evaluate it on the basis of our own tests and controlled versions running for several moths on different desktop computers.

Firstly, in order to work properly, *SpamStop* needs to be trained with enough data. Although, as discussed above, the general rule for Support Vector Machines states "the more data, the better the results", a critical number of training instances being sufficient to start with was empirically determined. Over all different training sets and all test runs, two hundred e-mails, about one hundred of each category, is enough in order for *SpamStop* to work to a satisfying degree. It is true however, that performance gets better with more training instances.

In its current version, *SpamStop* has a correctness performance of 90% to 95%, i.e. it reaches the same percentages as the leading products on the market. There are rarely false positive classifications. The remaining 10% to 5% are mainly very ambiguous e-mail like electronic greeting cards, which often include advertisements in the very same e-mail.

When trying to evaluate these numbers, one has to bear in mind that *SpamStop* does not yet include a variety of standard pre-filtering mechanisms, which should be included in a possible end-user product. This has not yet been done since it was the main goal of this study to assess the possibility of using a single SVM for filtering spam e-mail. So long, each incoming e-mail is examined in the same way. If standard pre-filtering methods were applied, clearly positive and negative e-mails would be removed beforehand, thus allowing *SpamStop* to focus on the grey area in-between. Since they are not incorporated, out of the variety of mechanisms, only some shall be briefly mentioned here.

It is popular to preprocess e-mail accordingly to its sender's address. Lists are maintained which name spam-senders on the one hand ("black list") and known persons on the other ("white list"). If the sender's address of an incoming mail is in either of the two lists, that e-mail is not further considered, but directly sorted into the corresponding category instead. The program architecture of *SpamStop* would even allow improving this technique further, using addresses in the training sets. With this trick, it could be possible to capture ambiguous cases as the mentioned example of the electronic greeting cards, given that there is at least one instance among the positive instances.

Another recent method to filter out a lot of undesired e-mails is to ask the server from which the corresponding e-mail apparently originated whether the sender's address really exists there.

Since *SpamStop* is up to now only a client based system, another aspect of how *SpamStop* could be improved, is to further develop it to incorporate it into networks. This could be easily fulfilled with a paradigm of local training and global filtering. The model used for the classification, could be applied to e-mail directly on the server. This would prevent a lot of unnecessary traffic caused by spam e-mails and would therefore be a valuable tool against the spam flood that is expected during the next years.

References

Carl, C. (2003). *Kernels for Structures*. Bachelor Thesis. University of Osnabrück

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Cristianini, N., & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines*. Cambridge University Press.

Hird, S. (2002). Technical Solutions for Controlling Spam. In *Proceedings of AUG2002*, Melbourne, 4-6 September, 2002.

Madigan, D. (2003). *Statistics and the War on Spam*. Online at http://www.stat.rutgers.edu/~madigan/PAPERS/sagtu.pdf (04-07-21)

T. Joachims (1998). Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Cline Rouveirol, editors, *Proceedings of the European Conference on Machine Learning*, pp. 137-142, Berlin, Springer.

SpamStop – Intelligent Neural Networks against Spam. Software available at http://www.spamstop.panmental.de