

# METHODS FOR DYNAMICALLY COMBINING RELEVANCIES OF DIFFERENT ANTISPAM FILTERS

Alexandru Catalin COSOI (acosoi@bitdefender.com)

*Researcher / BitDefender Antispam Laboratory  
5<sup>th</sup>, Fabrica de Glucoza Str, Bucharest, Romania, Phone: +40212330780*

**ABSTRACT:** Combining results from different AntiSpam filters has been a major concern for the AntiSpam community for many years. Questions like which filter should be more relevant in computing the output score, how high should the user decidability be, how to combine different results from different filters for a better computed spam indicator, what happens with the performance over time, troubled our minds since the beginning of spam. Different algorithms have been presented for computing these weights, but we believe that the answer to this problem is to calculate them dynamically, for each user individually and also to introduce new parameters that account for the time passed since the last update, or how long it takes until the filter starts to lose performance. We believe this way AntiSpam solutions will be able to have less false positives and a higher detection rate.

Keywords: spam, AntiSpam, relevance, filters, online and offline training, dynamical computation

## INTRODUCTION

This paper deals with AntiSpam solutions that make use of filters that are trained both offline and online. Since the first introduction of the Bayesian filtering methods, more than 80 percent of the current AntiSpam solutions available on market today make use of it. Of course, Bayesian filtering isn't the only known filtering method that can be trained online, but since it is the most popular, it will be used as an example in our presentation. One of the main concerns of spam analysts today is how to combine the results of several filters to achieve accuracy. Usually, the relevancies of these filters are computed offline and then the result is sent to the user.

However, in the case of Bayesian filtering methods for instance, this practice might not be the best. There is no certainty that computing offline the relevancies for two filters, trained one at the user and the other one at the developer is better than computing them online.

In this paper we shall define a set of parameters for an abstract filter. We will then show how these parameters can be used to compute the relative relevancies of filters in a set.

The parameters in question are expressions of facts such as: what is the (known) risk of a certain filter producing a false positive, what is the "maturity" level of the filter (in terms of adequacy of

training) and so on.

The method to compute relative relevancies we devised is based on balancing the relevance values of the user trained filters and the developer trained filters in such a way that whenever updates are delayed or detection rates are low the “majority vote” is given to the user-trained filters (if they were indeed well-trained previously – we call this factor the “maturity” of a filter) and vice versa.

## THE APPROACH

Several methods for an initial relevance computation have been proposed, like support vector machines, ROC curves, statistical methods, etc, but none of them can be adjusted after deployment. Therefore, these should be complemented by a more flexible method, such as the one we are proposing.

In the last two years of analysis on different AntiSpam solutions (even our own) we noticed that in time a discrepancy between the results of the user-trained filters and the developer-trained filters is created.

If the filters aren't updated for a few days (or weeks), the detection rate decreases, even if the user continues to train his filter very efficiently on the type of spam that arrives at their particular location.

We found that in most cases this problem was caused by a flaw in the formula that calculates the final score. Of course, from a user point of view, one solution would be to deactivate the offline-trained filters and to use his personal one, but this would not work in the case of a flood of “new” spam. We believe to have found a new approach to this problem. We have developed a mathematical system that balances the relevance between the users developed filter and the filters updated by the software company.

The principle is quite easy to exemplify. Let's consider a company that produces AntiSpam software. Its product contains four filters: a Bayesian one, which is delivered empty (untrained) for the user to train at will, a heuristic filter, which is delivered already trained and has to be updated frequently and two other indifferent filters that are also trained by the software producers. These filters “vote” on mail and an aggregate score is generated, usually by summing the output scores of various filters after applying some sort of relevance factor to each. The e-mail is then tagged as “spam” if the total score passes a certain threshold or as “legit” if it does not.

The following weighted average is generally used to compute the output score:

$$S = \frac{HR * Hr + BR * Br + F1R * F1r + F2R * F2r}{Hr + Br + F1r + F2r}$$

Where

- S – the total score an email can achieve (scaled between 1 and 100)
- HR – the result returned by the heuristic filter
- BR – the result returned by the Bayesian filter
- F1R – the result returned by the first unknown filter
- F2R – the result returned by the second unknown filter
- Hr – the relevance of the heuristic filter
- Br – the relevance of the Bayesian filter
- F1r – the relevance of the first unknown filter
- F2r – the relevance of the second unknown filter

HR, BR, F1R, F2R are computed using some statistical algorithm (ROC curves, SVM, GA, etc). Each of these parameters determines the importance of each filter in the decision process. This formula can be further improved by considering that whenever a filter returns a “I don’t know” value, which means that the filter can’t decide if it’s spam or legitimate, its relevance should be proportionally assigned to the other filters that return a possibly conclusive value (“runoff”).

This is a very good approach, but the relevancies are modified only in cases when one or more filters cannot decide. This can be caused either by ambiguous content or by a relatively new type of spam (portions of it or the entire message were never seen before during the training phase).

We believe that when computing the relevance for each filter, extra information should be used. For instance if we believe that we have an exhaustive training corpus for a certain filter, this belief should reflect in the importance assigned to that filter. In the following lines you will see the extra parameters that we believe that should be part in the decision process when computing the relevancies.

- M – The maturity reached by the Bayesian filter (scaled between 1 and 100 or 0 and 1). After a certain number of emails have entered the training phase of the Bayesian filter it is usually considered to be trained and mature. But let’s consider the following example. Whenever a false negative arrives at the filter, the user should press “This is spam”. Until now, since we actually add another email to the database of the Bayesian filter, its maturity score is increased, even though the e-mail was misclassified at first. If we repeat this many times, we will soon see some major misclassifications performed by this filter, because the filter gains in relevance while still not producing accurate results. The maturity of the filter should be increased only in the case of success. This means that a correction module should be integrated with “Yes, that was spam”, and “No, that wasn’t spam”, and some automatic answers. For example, in case the user never pushed the “Yes that was spam” and still a certain e-mail was tagged as spam and ignored for a couple of days or deleted, that means it was really spam so we can increase the maturity of the filter.
- T - Parameter which indicates how old the updated information is. Clearly, after an entire year without updates, using the filters would be worse than useless. So, we will use a value from 0 to 365 (scaled between 0 and 1 or 0 and 100) to denote the age of the filter. Of course,

if the company that produces that particular AntiSpam filter is out of business and can't send any more updates or we have a user that wasn't very interested in renewing the license for the filters developed by that company, we might as well disconnect those filters after 365 days since the last update.

- A – Parameter which indicates the aging property of the filter. Statistically we can determine what happens with the performance of each filter on a changing corpus. It is very important to know what happens the next day, or the third day with the detection rate after a filter update. Some filters can keep their detection rate high even after a few weeks without requiring to be updated; others should be updated daily or hourly. The direct relation between A and T is obvious. A filter that has a high aging value and was updated today should have today a higher relevance in the decision process. Tomorrow, in concordance with T, the relevance should be a bit smaller.
- FP and FN – every filter should have a false positive and false negative risk assigned. If a filter is known to have by itself some false positives this should also be reflected when computing its relevance.
- C – Short-term experiments and common sense usually indicate that a filter is more accurate than another, although statistically things are vice versa. If a thousand spam messages have the word “mother” in them, and a certain filter uses it for short term detection, it doesn't mean that it has amazing results and this word is a good heuristic to consider in the detection process. This parameter indicates the trust we have in this filter's answer.

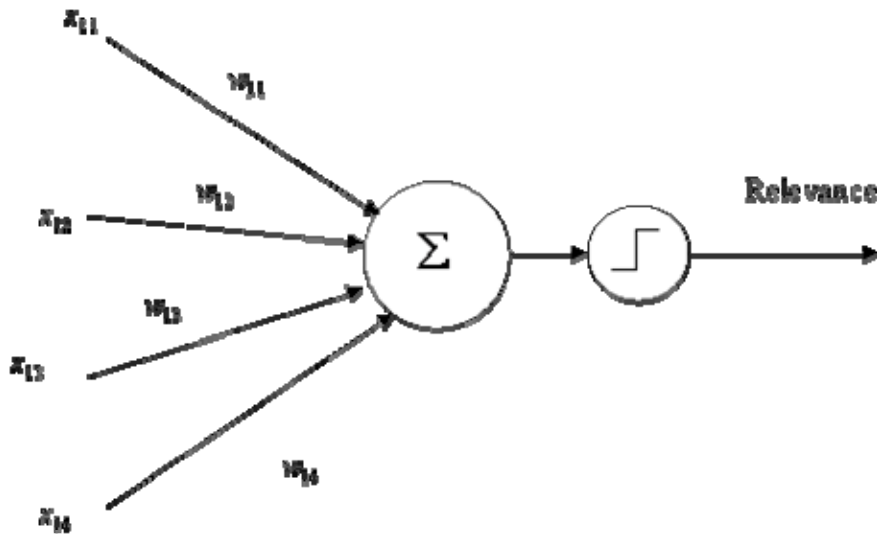
## ALGORITHMS

There are several methods that can incorporate these parameters when computing the relevancies. We're not trying to reinvent the wheel, but rather to point out that there should be some extra information involved when computing a filter's relevance.

The first step should be deciding what parameters are involved when computing this filter's relevance. It is obvious that yes or no filters have a special treatment, totally different from filters which return a score. User-trained filters can't have a T parameter, or at least not one with the same meaning as a company trained filter, because trainings performed on this filter are not iterated on an entire new corpus, but rather on an exclusive corpus gathered by the user in his personal mailbox(es).

After this process is performed, we can use several methods for computing the relevance of a filter. For instance, we could use a neural network to map our input information to the desired relevance. We consider this a good idea since neural networks are known for their ability to generalize information, although the process itself is not so transparent.

Each filter is assigned a single neuron which receives the parameters described above and returns the relevance for that filter. This result which is scaled between 0 and 1 should then be rescaled in accordance to the other relevancies computed with the same method so that the following condition is maintained:



In the drawing above a single perceptron is presented. As inputs, it receives the parameters described in this paper. As desired output it will be assigned the desired optimal relevance, pre-computed using classical methods in real life situations. After a number of training cycles, the weights of this perceptron will carry enough information to accurately predict the best relevance for each filter in various situations.

As you may notice, training these perceptrons would be the only difficult part. In our experiments, we had to simulate a few virtual users, some that train more often, some that train rarely, and even some that don't train their Bayesian filters at all in order to obtain data for the perceptrons. Every day we computed our filters' relevancies using ROC curves and then fed in the perceptrons this information for learning. After two weeks, the perceptrons started to predict the relevance with a high success rate.

Another method would be to use the following formula:

$$R_i = M_i - T_i \left[ A_i + \frac{C_i(FN_i + FP_i) + 2FN_iFP_i}{C_i^2 + C_i(FN_i + FP_i) + FN_iFP_i} \right]$$

Where

- $R_i$  represents the relevance for the  $i$ 'th filter on a scale from 0 to 100
- $C_i$  represents our confidence in that filter on a scale from 0 to 100
- $M_i$  represents the maturity level that filter achieved on a scale from 0 to 100
- $T_i$  represents the filter's age, where young means that the filter was updated recently and old means that it was updated long time ago. Scaled between 0 and 100, it indicates that we

should decrease the filter's relevance every time it ages.

- $A_i$  is the aging factor
- $FP_i$  and  $FN_i$  are the false positive risk and the false negative risk for the  $i$ 'th filter.

For a better presentation of this formula let's consider the following two examples:

M = 98 %	M = 98 %
T = 0.2 %	T = 4 %
A = 0.01 %	A = 0.01 %
FP = 1.26 %	FP = 1.26 %
FN = 9 %	FN = 9 %
C = 90 %	C = 90 %
R = 97.978 %	R = 97.56 %

As you can see, as time passes (time since the last update for that filter), relevance decreases. In just 10 days since the last update, although this is a good filter judging by the described parameters, it loses almost 0.5% importance in the decision process.

The formula above can also be presented in the following way:

$$R_i = M_i - T_i(A_i + \Omega_{i1} + \Omega_{i2}), \text{ where}$$

$$\Omega_{i1} = 1 - \frac{C}{C + FP}$$

$$\Omega_{i2} = 1 - \frac{C}{C + FN}$$

Of course, there can be several other methods for computing the relevance using these parameters.

Although it's not that easy to train these perceptrons, we think using this method is superior to using the mathematical equation, although implementing the latter is trivial. The perceptron's ability to generalize information would be a high advantage when dealing with unforeseen examples.

After implementing a relevancy decision process, a module which takes the output of each neuron or of each formula and establishes proportional relevancies between the filters should be integrated.

This module would transform the relevancies in percents, in order to match the

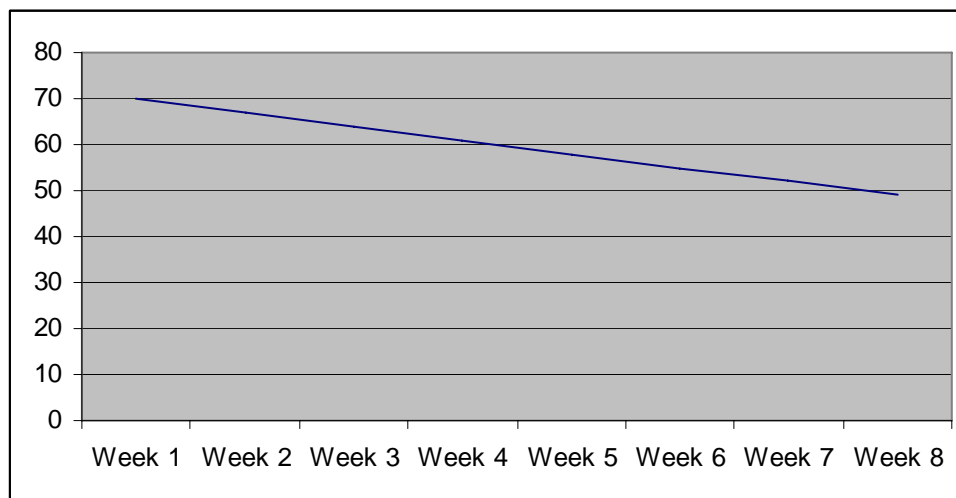
condition  $\sum_{i=1}^{NumFilters} Relevance_i = 1$ , or, scaled between 0 and 100:  $\sum_{i=1}^{NumFilters} Relevance_i = 100$ .

The new relevancies will be  $r_i = \frac{100 \cdot R_i}{\sum_{j=1}^{num\_filters} R_j}$ .

## RESULTS

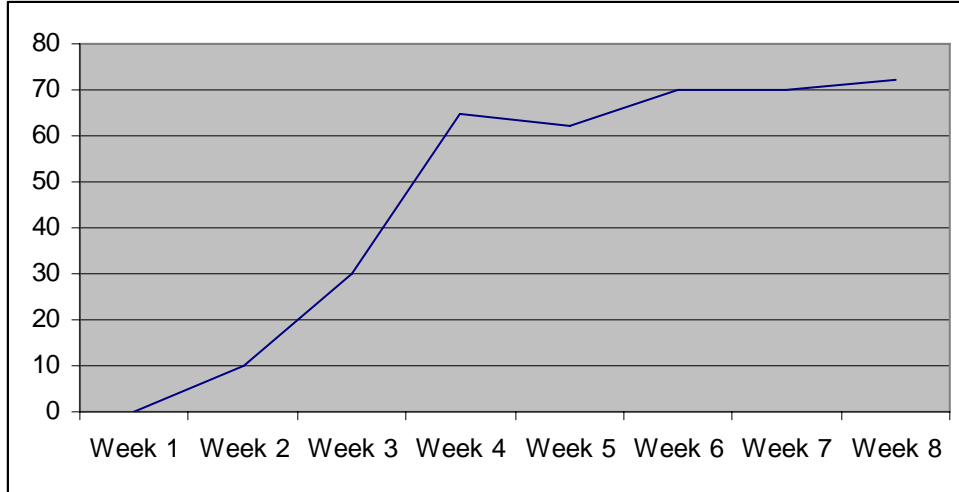
There are still a lot of tests that have to be done before this procedure will be used in the market although the preliminary results are quite interesting. We followed the evolution of a pre-trained Bayesian filter in a solution that incorporated a heuristic filter, a user-trained Bayesian filter (delivered initially empty to the client) and a pre-trained Bayesian filter. We simulated a user that was constantly training his Bayesian filter and we also modified the maturity level not only if the filter was trained by some minimum number of email messages, but even tried to apply a penalty/reward algorithm on that parameter.

The pre-trained Bayesian filter was delivered with a large database, and it was trained on a few million varied samples. That meant that its confidence and maturity were quite high in the beginning (but also the aging factor sadly). After two months of no updates, its relevance decreased from 70% down to 49%.



**Figure 1 - Relevance decreases for an untrained filter**

On the other hand, the user trained Bayesian had to start from 0 confidence and 0 maturity level. During the first week there were no results, since the user did not achieve the minimum number of email to train so that the filter could be activated. After two months the relevance grew to 72%.



**Figure 2 - Relevance increases for a constantly trained filter**

You can see in the graph that from week 4 to week 5 there's a small fall in the relevance. In that period the user did not train his filter, but rather he applied (rarely) the penalty module. (by selecting one of the two choices: "yes, it was spam", and "no, it wasn't spam").

As you can see, the relevance takes into consideration that if the company filter did not updated, more relevance should be assigned to the user trained filter (of course, with the condition that this filter was constantly trained too).

As you may know, the perceptron tries to accommodate inputs to the desired outputs. Of course, we won't need the exact perceptron theory applied here. For example, in theory, the value computed by summing the products of each input with the assigned weight is then compared with a threshold value, in order to determine if the neuron activates itself or not (if the neuron returns a value or not).

$$\sum_i x_i w_i \geq \theta$$

In our case, there is no threshold value. We will need to have a result all the time. Basically we try to determine a function  $f$  which takes as input the parameters defined in this paper and produces as output the desired relevance.

For example, for the  $i$ 'th filter we will have the following function:

$$f_i(M_i T_i, A_i T_i, C_i T_i, FPT_i, FN_i T_i) = R_i$$

Or if we want to minimize the number of parameters, we could try

$$f_i(M_i T_i, A_i T_i, \Omega_{1i} T_i, \Omega_{2i} T_i) = R_i$$

If we would want our results to be really close to the formula presented above, we could use this definition of the formula:

$$f_i(M_i, -A_i T_i, -\Omega_{1i} T_i, -\Omega_{2i} T_i) = R_i$$

If during training, we would feed the neuron with the outputs computed with our formula, it is obvious that the weights will be really close to 1.

The training algorithm is almost the same presented by Frank Rosenblatt in 1957, with minor differences. The purpose of the training is to modify the weights in order to obtain the desired relevancies.

We tried all three types of function definition in order to sustain our approach, and the final results we're similar. Also, we tried training on a small time interval and tested on parameters computed weeks latter. The results were quite accurate. (an error of 0.02 between the relevance computed then and the relevance we computed a week earlier).

As an example, for the filter with the parameters presented in the table above, here's what happens with the relevance in a time frame of 8 weeks.

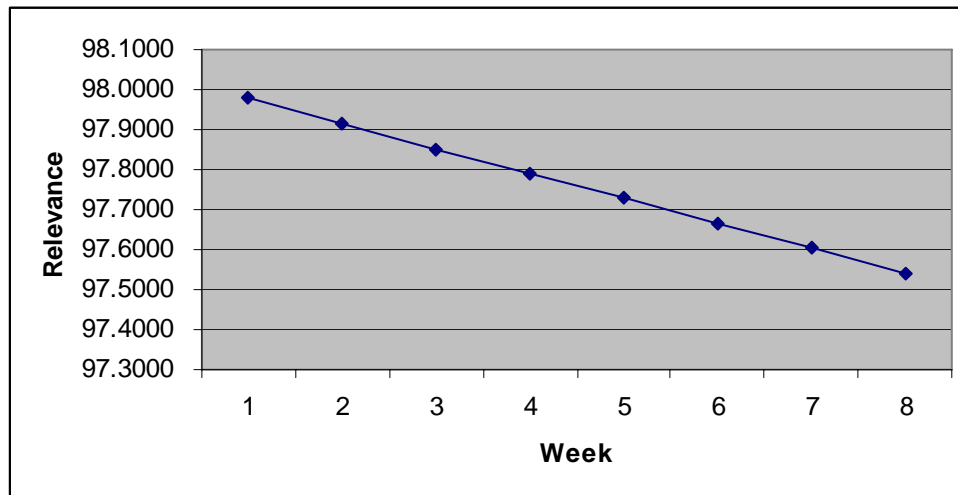


Figure 3 - Relevance computed with a modified perceptron during 8 weeks

As you may notice, the results are quite similar with the ones computed with our formula.

## CONCLUSIONS

We believe that this approach in computing the relevancies of several AntiSpam filters is a good idea. Spam changes every day, some filters are easily updated, some aren't. Balancing the relevancies between the user's trained filters and the company trained filters, modifying the importance of each filter based on time, confidence, maturity, known risks and aging factor are all important parameters in the decision process.

Each filter's relevance should not be fixed by the software producers, but rather computed for each user individually, with the user's personalized information as a basis for decisions.

## BIBLIOGRAPHY

1. Interpreting diagnostics tests – Thomas G. Tape, MD, University of Nebraska
2. Support vector machines and other kernel-based learning methods – John Shawe-Taylor, Nello Cristianini, Cambridge University Press
3. The perceptron: A perceiving and recognizing automaton - Rosenblatt, F., Technical Report 85-460-1, Cornell Aeronautical Laboratory.
4. *Principles of Neurodynamics* – Rosenblatt F., Spartan Books, New York
5. Genetic algorithms in Search, Optimization, and Machine Learning – David E. Goldberg
6. An AntiSpam filter based on adaptive neural networks – Catalin Cosoi, SpamConference 2006
7. An introduction to Genetic Algorithms – Melanie Mitchell