Boosting Trees for Anti-Spam Email Filtering [Extended Version]

Xavier Carreras Lluís Màrquez

TALP Research Center
LSI Department
Universitat Politècnica de Catalunya (UPC)
Jordi Girona Salgado 1–3
Barcelona E-08034, Catalonia
{carreras,lluism}@lsi.upc.es

September 2001

Abstract

In this work, a set of comparative experiments for the problem of automatically filtering unwanted electronic mail messages are performed on two public corpora: PU1 and LingSpam. Several variants of the AdaBoost algorithm with confidence–rated predictions [14] have been applied, which differ in the complexity of the base learners considered. Two main conclusions can be drawn from our experiments: a) The boosting–based methods clearly outperform the other learning algorithms results published on the two evaluation corpora, achieving very high levels of the F_1 measure; b) Increasing the complexity of the base learners allows to obtain better "high–precision" classifiers, which is a very important issue when misclassification costs are considered.

1 Introduction

Spam-mail filtering is the problem of automatically filtering unwanted electronic mail messages. The term "spam mail" is also commonly referred to as "junk mail" or "unsolicited commercial mail". Nowadays, the problem has achieved a big impact since bulk emailers take advantage of the great popularity of the electronic mail communication channel for indiscriminately flooding email accounts with unwanted advertisements. The major factors that contribute to the proliferation of unsolicited spam email are the following two: 1) bulk email is inexpensive to send, and 2) pseudonyms are inexpensive to obtain [4]. On the contrary, individuals may waste a large amount of time transferring unwanted messages to their computers and sorting through those messages once transferred, to the point that they may be likely to become overwhelmed by spam.

Automatic IR methods are well suited for addressing this problem, since spam messages can be distinguished from the "legitimate" email messages because of their particular form, vocabulary, and word patterns, which can be found in the header or body of the messages.

The spam filtering problem can be seen as a particular instance of the Text Categorization problem (TC), in which only two classes are possible: *spam* and *legitimate*. However, since one is the opposite of the other, it also can be seen as the problem of identifying a single class, *spam*. In this way, the evaluation of automatic spam filtering systems can be done by using common measures of IR (precision, recall, etc.). Another important issue is the relative importance between the two types of possible misclassifications: While an automated filter that misses a small percentage of spam may be acceptable to most users, fewer people are likely to accept a filter that incorrectly identifies a small percentage of legitimate mail as spam, especially if this implies the automatic discarding of the misclassifed legitimate messages. This problem suggests the consideration of misclassification costs for the learning and evaluation of spam filter systems.

In recent years, a vast amount of techniques have been applied to TC, achieving impressive performances in some cases. Some of the top–performing methods are Ensembles of Decision Trees [17], Support Vector Machines [8], Boosting [15] and Instance–based Learning [19].

Spam filtering has also been treated as a particular case of TC. Cohen [3] used a method based on TF-IDF weighting and the rule learning algorithm RIPPER to classify and filter email. Sahami et al. [12] used the Naive Bayes approach to filter spam email. Drucker et al. [5] compared Support Vector Machines (SVM), boosting of C4.5 trees, RIPPER and Rocchio, concluding that SVM's and boosting are the top–performing methods and suggesting that SVM's are slightly better in distinguishing the two types of misclassification. Androutsopoulos and colleagues compared Sahami's Naive Bayes against the TiMBL Memory-based learner [2], and combined these two approaches in a stack of classifiers which improved the performance [13]. In their work, these autors introduce cost-sensitive evaluation measures and two public data sets, the PU1 [1] and the LingSpam [2], which might become standard benchmark corpora for the problem.

In this paper, we show that the AdaBoost algorithm with confidence–rated predictions is a very well suited algorithm for addressing the spam filtering problem. We have obtained very accurate classifiers on two corpora, namely PU1 and LingSpam, and we have observed that the algorithm is very robust to overfitting. Another advantage of using AdaBoost is that no prior feature filtering is needed since it is able to efficiently manage large feature sets (of tens of thousands).

In the second part of the paper we show how increasing the expressiveness of the base learners can be exploited for obtaining the "high–precision" filters that are needed for real user applications. We have evaluated the results of such filters using the measures introduced in [1], which take into account the misclassification costs, and have substantially improved the results mentioned in that work.

The paper is organized as follows: Section 2 is devoted to explain the AdaBoost learning algorithm and the variants used in the comparative experiments. In section 3 the setting is presented in detail, including the corpora and the experimental methodology used. Section 4 reports the experiments carried out and the results obtained. Finally, section 5 concludes and section 6 outlines some lines for further research.

2 The AdaBoost Algorithm

In this section the generalized AdaBoost algorithm with confidence-rated predictions [14] is described, restricting to the case of binary classification.

The purpose of boosting is to find a highly accurate classification rule by combining many *weak rules* (or weak hypotheses), each of which may be only moderately accurate. It is assumed the existence of a separate procedure called the WeakLearner for acquiring the weak hypotheses. The boosting algorithm finds a set of weak hypotheses by calling the weak learner repeatedly in a series of T rounds. These weak hypotheses are then linearly combined into a single rule called the *combined hypothesis*.

Algorithm 1: The AdaBoost learning algorithm

AdaBoost(**in:** $S = \{(x_i, y_i)\}_{i=1}^m$)

  for $i = 1$ **to** m

    $D_1(i) \leftarrow 1/m$

  for $t = 1$ **to** T

    $h_t \leftarrow$ WeakLearner(S, D_t)

    Choose $\alpha_t \in \mathbb{R}$

    for $i = 1$ **to** m

      $D_{t+1}(i) \leftarrow \dfrac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

      # Z_t is chosen so that D_{t+1} will be a distribution

  return the combined hypothesis: $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$

Let $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ be the set of m training examples, where each instance x_i belongs to a instance space $\mathcal{X}$ and $y_i \in \{-1, +1\}$ is the class or label associated to x_i. The goal of the learning procedure is to produce a function of the form $f : \mathcal{X} \rightarrow \mathbb{R}$, such that, for any example x, the sign of $f(x)$ is interpreted as the predicted class (-1 or $+1$), and the magnitude $|f(x)|$ is interpreted as a measure of confidence in the prediction. Such a function can be used either for classifying or ranking new unseen examples.

The pseudo–code of AdaBoost is presented in Algorithm 1. It maintains a vector of weights as a distribution D over examples. The goal of the WeakLearner algorithm is to find a weak hypothesis with moderately low error with respect to these weights. Initially, the distribution D_1 is uniform, but the boosting algorithm exponentially updates the weights on each round to force the weak learner to concentrate on the examples which are hardest to predict by the preceding hypotheses.

More precisely, let D_t be the distribution at round t, and $h_t : \mathcal{X} \rightarrow \mathbb{R}$ the weak rule acquired according to D_t. In this setting, weak hypotheses $h_t(x)$ also make real–valued confidence–rated predictions (i.e., the sign of $h_t(x)$ is the predicted class, and $|h_t(x)|$ is interpreted as a measure of confidence in the prediction). A parameter α_t is then chosen and the distribution D_t is updated. The choice of α_t will be determined by the type of weak learner (see next section). In the typical case that α_t is positive, the updating function decreases (or increases) the weights $D_t(i)$ for which h_t makes a good (or bad) prediction, and this variation is proportional to the confidence $|h_t(x_i)|$.

The final hypothesis, f, computes its predictions using a weighted vote of the weak hypotheses.

In [14] it is proven that the training error of the AdaBoost algorithm (i.e. the fraction of training examples i for which the sign of $f(x_i)$ differs from y_i) is at most $\prod_{t=1}^{T} Z_t$, where Z_t is the normalization factor computed on round t. This upper bound is used in guiding the design of both the parameter α_t and the WeakLearner algorithm, which attempts to find a weak hypothesis h_t that minimizes Z_t.

2.1 Learning Weak Rules

In [14] three different variants of AdaBoost are defined, corresponding to three different methods for choosing the α_t values and calculating the predictions of the weak hypotheses. In this work we concentrate on AdaBoost *with real–valued predictions* since it is the one that has achieved the best results in the Text Categorization domain [15].

According to this setting, weak hypotheses are simple rules with real–valued predictions. Such simple rules test the value of a boolean predicate and make a prediction based on that value. The predicates used refer to the presence of a certain word in the text, e.g. "the word *money* appears in the message". Formally, based on a given predicate p, our interest lies on weak hypotheses h which make predictions of the form:

$$h(x) = \begin{cases} c_0 & \text{if } p \text{ holds in } x \\ c_1 & \text{otherwise} \end{cases}$$

where the c_0 and c_1 are real numbers.

For a given predicate p, the values c_0 and c_1 are calculated as follows. Let X_1 be the subset of examples for which the predicate p holds and let X_0 be the subset of examples for which the predicate p does not hold. Let $[\![\pi]\!]$, for any predicate π, be 1 if π holds and 0 otherwise. Given the current distribution D_t, the following real numbers are calculated for $j \in \{0, 1\}$, and for $b \in \{+1, -1\}$:

$$W_b^j = \sum_{i=1}^{m} D_t(i)[\![x_i \in X_j \wedge y_i = b]\!] \,.$$

That is, W_b^j is the weight, with respect to the distribution D_t, of the training examples in partition X_j which are of class b. As it is shown in [14]

Z_t is minimized for a particular predicate by choosing:

$$c_j = \frac{1}{2} \ln \left(\frac{W^j_{+1}}{W^j_{-1}} \right) . \tag{1}$$

and by setting $\alpha_t = 1$. These settings imply that:

$$Z_t = 2 \sum_{j \in \{0,1\}} \sqrt{W^j_{+1} W^j_{-1}} \, . \tag{2}$$

Thus, the predicate p chosen is that for which the value of Z_t is smallest.

Very small or zero values for the parameters W^j_b cause c_j predictions to be large or infinite in magnitude. In practice, such large predictions may cause numerical problems to the algorithm, and seem to increase the tendency to overfit. As suggested in [15], the following smoothed values for c_j have been considered:

$$c_j = \frac{1}{2} \ln \left(\frac{W^j_{+1} + \epsilon}{W^j_{-1} + \epsilon} \right) . \tag{3}$$

giving ϵ the value $1/m$. This smoothing affects the value of Z as follows:

$$Z_t = \sum_{j \in \{0,1\}} \left(W^j_+ \sqrt{\frac{W^j_- + \epsilon}{W^j_+ + \epsilon}} + W^j_- \sqrt{\frac{W^j_+ + \epsilon}{W^j_- + \epsilon}} \right) . \tag{4}$$

It is important to see that the so far presented weak rules can be directly seen as decision trees with a single internal node (which tests the value of a boolean predicate) and two leaf nodes that give the real-valued predictions for the two possible outcomes of the test. These extremely simple decision trees are sometimes called *decision stumps*. In turn, the boolean predicates can be seen as binary features (we will use the word *feature* instead of *predicate* from now on). Thus, the already described criterion for finding the best weak rule, or the best feature, can be seen as a natural splitting criterion and used for performing decision–tree induction [14].

Following the idea suggested in [14] we have extended the WeakLearner algorithm to induce arbitrarily deep decision trees. The splitting criterion used consists in choosing the feature that minimizes equation (2), while the predictions at the leaves of the boosted trees are given by equation (1). Note that the general AdaBoost procedure remains unchanged.

In this paper, we will denote as TreeBoost the AdaBoost algorithm including the extended WeakLearner. TreeBoost[d] (or TB[d] in some figures) will stand for a learned classifier with weak rules of depth d. As a special case, TreeBoost[0] will be denoted as Stumps.

3 Setting

3.1 Domain of Application

The empirical experiments in this work have been performed on two public corpora for the anti-spam email filtering problem: the PU1 and the LingSpam.[1] The PU1 corpus contains personal messages and spam messages. The LingSpam corpus contains messages extracted from the Linguist list and spam messages. Table 3.1 shows the sizes of the two corpora. On the one hand, the percentage of spam emails on the LingSpam seems to be more realistic than the percentage on the PU1. On the other hand, the topics of legitimate messages in the PU1 are more diverse than in the LingSpam, in which all messages are related to linguistics.

The two corpora are presented partitioned into 10 folds, in order to perform the experiments using 10-fold cross-validation.

The feature set of the corpora is the commonly used in the text categorization literature bag-of-words model with binary features. In this model, each word in the vocabulary forms a binary feature, and a message is represented having the features corresponding to the occurring words set to true. The features of the PU1 corpus are given encrypted, in order to protect the privacy of the personal messages. Four versions of the corpora are available, differing on the use of two resources: a stop-word list and a lemmatizer. The enabling/disabling of these two resources in the generation results in the following four versions:

- BARE: No resources used. The features are the lower-cased words.
- LEMM: Lemmatizer used; no stop-word removal.
- STOP: Stop-word list used; no lemmatizer.
- LEMM STOP: Both lemmatizer and stop-word list used.

Table 3.1 shows the feature space dimension of the four versions.

3.2 Experimental Methodology

Evaluation Measures. The measures used in this paper for evaluating the spam filtering system are introduced in this section. Let S and L be the number of spam and legitimate messages in the corpus, respectively; let

[1] The two corpora are freely available from the following address `http://www.iit.demokritos.gr/~ionandr`

	PU1	Lingspam
Messages	1,099	2,893
Spam	481	481
Legitimate	618	2,412
Spam %	43.7%	16.6%
BARE size	26,449	65,723
LEMM size	23,312	NA
STOP size	26,275	NA
LEMM STOP size	23,137	NA

Table 1: Dimensions of the evaluation corpora.

S_+ denote the number of spam messages that are correctly classified by a system, and S_- the number of spam messages misclassified as legitimate. In the same way, let L_+ and L_- be the number of legitimate messages classified by a system as spam and legitimate, respectively. These four values form a contingency table which summarizes the behaviour of a system. The widely-used measures precision (p), recall (r) and F_β are defined as follows:

$$p = \frac{S_+}{S_+ + L_+} \qquad r = \frac{S_+}{S_+ + S_-} \qquad F_\beta = \frac{(\beta^2+1)pr}{\beta^2 p + r}$$

The F_β measure is the harmonic mean between precision and recall, and with $\beta = 1$ gives equal weigth to the combined measures. Additionally, some experiments in the paper will also consider the accuracy measure ($acc = \frac{L_- + S_+}{L+S}$).

A way to distinguish the two types of misclassification is the use of utility measures [9] used in the TREC evaluations [11]. In this general measure, positions in the contingency table are associated loss values, λ_{S+}, λ_{S-}, λ_{L+}, λ_{L-}, which indicate how desirable are the outcomes, according to a user-defined scenario. The overall performance of a system in terms of the utility is $S_+\lambda_{S+} + S_-\lambda_{S-} + L_+\lambda_{L+} + L_-\lambda_{L-}$.

Androutsopoulos et al. [1] propose particular scenarios in which misclassifying a legitimate message as spam is λ times more costly than the symmetric misclassification. In terms of utility, these scenarios can be translated into $\lambda_{S+} = 0$, $\lambda_{S-} = -1$, $\lambda_{L+} = -\lambda$ and $\lambda_{L-} = 0$. They also introduce the *weighted accuracy* (WAcc) measure, which is a λ-cost sensitive accuracy measure:

$$WAcc = \frac{\lambda \cdot L_- + S_+}{\lambda \cdot L + S}$$

When evaluating filtering systems, this measure suffers from the same problems as standard accuracy [18]. Despite this fact, we will use it for comparison purposes.

3.3 Baseline Algorithms

In order to compare our boosting methods against other techniques, we include the results of the following algorithms on the two evaluation corpora:

- **Decision Trees.** Standard TDIDT learning algorithm, using the RLM distance-based function for feature selection. We have evaluated it on the PU1 corpus. See [10] for complete details about the particular implementation.

- **Naive Bayes.** We include the best reported results obtained by the naive bayes approach on the PU1 Corpus [1] and on the LingSpam [2].

- ***k*-NN.** We include the best results on the LingSpam corpus reported in [2].

- **Stacking.** This approach combines Naive Bayes and k-NN in a stack of classifiers. Several configurations are evaluated on the LingSpam corpus in [13]; we report the best results of that paper.

3.4 Tuning Procedure

This section presents a general procedure for empirically tuning a parameter of the system, according to a given optimization criterion. The tuning procedure is designed for being part of the 10-fold cross-validation process. Given a parameter p to be tuned, an *evaluation measure* and an *optimization criterion*, the procedure is the following:

1. For each trial in the 10-fold cross-validation process:

   - Use 8 of the 9 training subsets for learning a classifier. We select the 8 folds randomly. The remaining fold is used as a validation subset.

   - For a range of values of the parameter p, test the classifier on the validation subset.

2. For each tested value of p, compute the *evaluation measure* using the outputs of all the trials.

3. According to the *optimization criterion*, select the best value for p as the value to be used in the final system.

4 Experiments

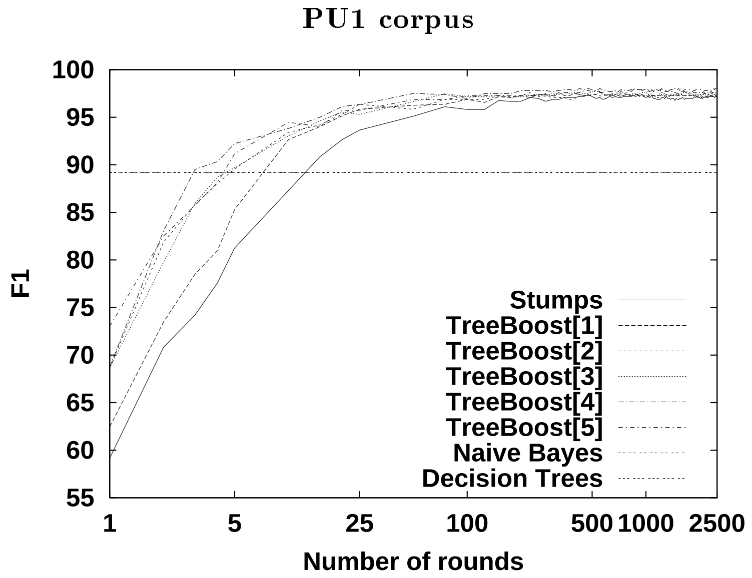
4.1 Comparing methods

The purpose of our first experiment is to show the general performance of AdaBoost in the spam-filtering domain. Six AdaBoost classifiers have been learned, setting the depth of the weak rules from 0 to 5; we denote each classifier as TreeBoost[d], where d stands for the depth of the weak rules; as a particular case, we denote the TreeBoost[0] classifier as Stumps. Each version of TreeBoost has been learned for up to 2,500 weak rules on the PU1 and 1,000 weak rules on the LingSpam.

Figure 1 shows the F_1 measure of each classifier learned, as a function of the number of rounds used. These plots also contain the performance of the baseline algorithms. It can be seen that TreeBoost clearly outperforms the other algorithms. The experiment also shows that, above a certain number of rounds, all TreeBoost versions achieve consistent good results, and that there is no overfitting in the process. After 150 rounds of boosting, all versions reach an F_1 value above 97% on the two corpora. It can be noticed that the deeper the weak rules, the smaller the number of rounds needed to achieve good performance. This is not surprising, since deeper weak rules handle much more information. Additionally, the figure shows avobe the stabilization, increasing the number of rounds results only in slight variations of the F_1 measure.

A concrete value for the T parameter of the TreeBoost learning algorithm must be given, in order to obtain real classifiers and to be able to make comparisons between the different versions of TreeBoost and baseline methods. To our knowledge, it is still unclear what is the best way for choosing T. On the PU1, we have estimated the T parameter with the tuning procedure presented in section 3.4, giving T values from 1 to 2,500 in steps of 25 and selecting the value which maximizes the F_1 measure. Since the performance is very stable and the tuning procedure involves intense processing, we have fixed 1,000 rounds for the classifiers on the LingSpam corpus.

Tables 2 and 3 present the results of all classifiers on the two corpora. For each classifier, we include the number of rounds used, recall, precision, F_1 and the maximum F_1 achieved all over the rounds learned. According to the results, boosting classifiers clearly outperform the other algorithms. Naive Bayes, k-NN and the Stack achieve precision rates slightly lower than



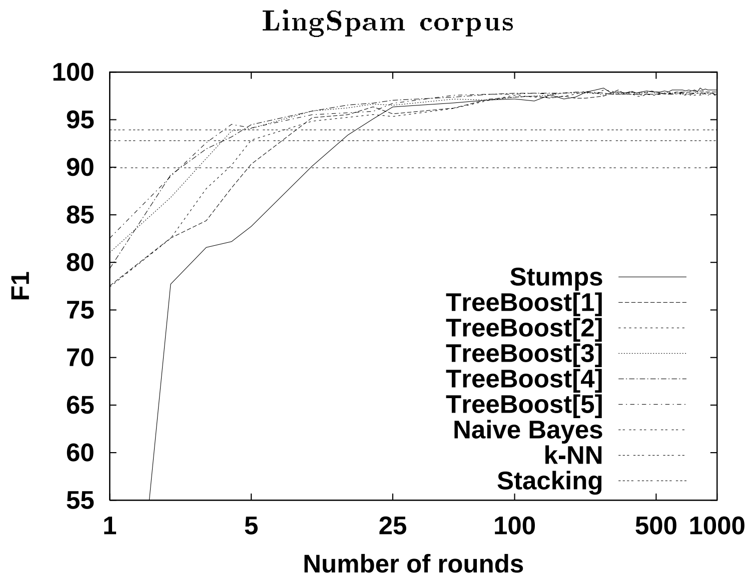


Figure 1: F_1 measure of Stumps and TreeBoost$[d]$, for increasing number of rounds

those obtained by boosting classifiers; however, the obtained recall rates are much lower.

	T	recall	precision	F_1	F_1^{max}
N. Bayes	-	83.98	95.11	89.19	-
D. Trees	-	89.81	88.71	89.25	-
Stumps	525	96.47	97.48	96.97	97.39
TreeBoost[1]	525	96.88	97.90	97.39	97.60
TreeBoost[2]	725	96.67	98.31	97.48	97.59
TreeBoost[3]	675	96.88	97.90	97.39	97.81
TreeBoost[4]	450	**97.09**	**98.73**	**97.90**	98.01
TreeBoost[5]	550	96.88	98.52	97.69	98.12

Table 2: Performance of all classifiers on the PU1 corpus

Accuracy results have been compared using the 10-fold cross-validated paired t test. Boosting classifiers perform significantly better than Decision Trees.[2] On the contrary, no significant differences can be observed between the different versions of TreeBoost. More interestingly, it can be noticed that accuracy and precision rates slightly increase with the expressiveness of the weak rules, and that this improvement does not affect the recall rate. This fact will be exploited in the following experiments.

4.2 High-Precision classifiers

This section is devoted to evaluate TreeBoost in high-precision scenarios, where only a very low (or null) proportion of *legitimate to spam* misclassifications is allowed.

4.2.1 Rejection Curves.

We start by evaluating whether the confidence of a prediction, i.e., the magnitude of the prediction, is a good indicator of the quality of the prediction or not. For this purpose, rejection curves are computed for each classifier. The procedure to compute a rejection curve is the following: For increasing proportions p between 0 and 100, reject the $p\%$ of the less confident predictions, both positive or negative, and compute the accuracy of the remaining $(100 - p)\%$ predictions. The desired behaviour is that accuracy values smoothly increase as long as p increases.

[2]Since we do not own the other classifiers, no tests have been ran; but presumably boosting methods are also significantly better.

	T	recall	precision	F_1	F_1^{max}
N. Bayes	-	82.40	99.00	89.94	-
k-NN	-	88.60	97.40	92.79	-
Stacking	-	91.70	96.50	93.93	-
Stumps	1,000	**97.92**	98.33	**98.12**	98.33
TreeBoost[1]	1,000	97.30	98.53	97.91	98.12
TreeBoost[2]	1,000	96.67	98.52	97.59	98.11
TreeBoost[3]	1,000	96.47	98.93	97.68	98.01
TreeBoost[4]	1,000	96.26	**99.14**	97.68	97.89
TreeBoost[5]	1,000	96.26	**99.14**	97.68	97.90

Table 3: Performance of all classifiers on the LingSpam corpus

Figure 2 plots the rejection curves computed for the six learned classifiers. Three curves are given in each plot: the accuracy curve, noted as 'All', the curve considering only misclassifications of *spam as legitimate*, noted as 'S to L', and the curve considering the misclassifications of *legitimate as spam*, noted as 'L to S'. The latter are the undisired type of errors of these scenarios. The following conclusions can be drawn:

- The confidence of a prediction is a good indicator of its quality. The higher the confidence of the predictions considered, the higher the accuracy achieved.

- The depth of weak rules greatly improves the quality of the predictions. On the PU1 corpus, whereas Stumps needs to reject the 73% of the less confident examples to achieve a 100% of accuracy, TreeBoost[5] only needs 23%. A similar but less significant behaviour occurs on the LingSpam. In other words, deeper TreeBoost filters concentrate the misclassified examples closer to the decision threshold.

- Spam predictions seem to be better than legitimate predictions, since the errors produced by the most confident predictions are of the type 'S to L'. A possible explanation is that the classifiers are biased to predict the most frequent class, that is legitimate. However, the results are not significant enough for stating this idea.

- A potential final email filtering application could have the following specification: Messages whose confidence of the prediction is greater than a threshold τ are automatically classified, i.e., spam messages are 'blocked' and legitimate messages are delivered to the user. Messages whose prediction confidence is lower than τ are stored in a special fold for dubious messages. The user has to verify if these are legitimate messages. This specification is suitable for having automatic filters with different degrees of strictness (i.e., different values for the τ parameter). τ values could be tuned using the procedure of section 3.4.

4.2.2 Cost–Sensitive Evaluation.

In this section, TreeBoost classifiers are evaluated using the λ-cost measures introduced in section 3. Three scenarios of strictness are presented in [1]: a) No cost considered, corresponding to $\lambda = 1$; b) Semi-automatic scenario, for a moderately accurate filter, giving $\lambda = 9$; and c) Completely automatic scenario, for a very high accurate filter, assigning $\lambda = 999$. As noted in section 3, we will consider these scenarios as particular utility matrices.

In [16] a simple modification of the AdaBoost algorithm for handling general utility matrices is presented. The idea is to initialize the weight distribution of examples according to the given utility matrix, and then run the learning algorithm as usual. We have performed experiments with this setting, but the results are not convincing: only the initial rounds of boosting are affected by the initialization based on utility; after a number of rounds, the performance seems to be like if no utility had been considered. Since our procedure for tuning the number of rounds can not determine when the initial stage ends, we have rejected this approach. We think that an appropiate modification of the AdaBoost algorithm should also consider the weight updating function.

Another approach consists in adjusting the decision threshold θ. In a default scenario, corresponding to $\lambda = 1$, an example is classified as spam if its prediction is greater than 0; in this case, $\theta = 0$. Increasing the value of θ results in a higher precision classifier. Lewis presented [9] a procedure for calculating the optimal decision threshold for a system, given an arbitrary utility matrix. The procedure is valid only when the system outputs probabilities, so the prediction margins resulting from the boosting classifications should be mapped into probabilities. A method for estimating probabilities given the output of AdaBoost is suggested in [7], using a logistic function. Initial experiments with this function have not worked properly, because relatively low predictions are sent to extreme probability values. A possible solution would be to scale down the predictions before applying the probability estimate; however, it can be observed that prediction margins grow

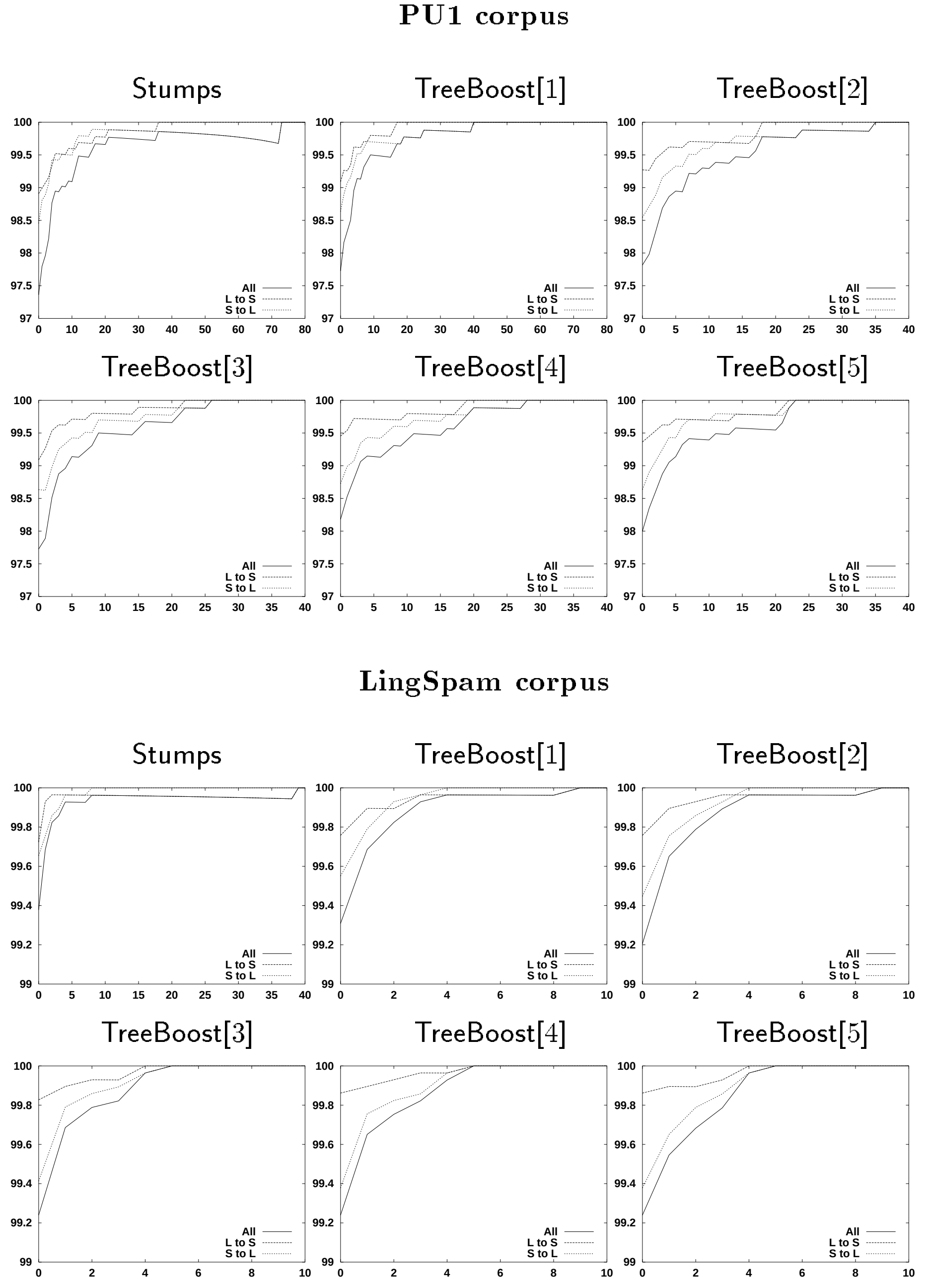


Figure 2: Rejection curves for all TreeBoost classifiers. x axis: percentage of rejected predictions; y axis: accuracy. Note that axis ranges are different in the plots.

with both the number and the depth of the used weak rules. Since many parameters are involved in this scaling, we have rejected the probability estimation of predictions.

Alternatively, we make our classification scheme sensitive to the λ factor by tuning the θ parameter to the value which maximizes the weighted accuracy measure. Once more, the concrete value for θ is obtained using the tuning procedure of section 3.4, in which several values for the parameter are tested. Tables 4 and 5 summarize the results obtained on the two corpus, giving λ factor values of 9 and 999. Results presented in [1] for the PU1 corpus, and [2] and in [13] for the LingSpam corpus are also reported.

Again, TreeBoost clearly outperforms the baseline methods. On the PU1 corpus, with $\lambda = 9$, very high-precision rates are achieved, maintaining considerably high recall rates. It seems that the depth of TreeBoost slightly improves the performance, although no significant differences can be achieved. For $\lambda = 999$, precision rates of 100% (which is the implicit goal in this scenario) are achieved, except for Stumps, maintaining fair levels of recall. However, recall rates are slightly unstable with respect to the depth of TreeBoost varying from 64.45% to 76.30%. On the LingSpam corpus, with $\lambda = 9$, TreeBoost classifiers achieve very high-precision rates and recall rates around 95% which clearly outperform the other methods. For $\lambda = 999$ very high-precision rates are achieved, but the recall rates are unstable. In this corpus, the differences in performance with respect to the depth of the weak rules are also unstable, and not convincing enough for stating conclusions.

Our impression is that high values in the λ factor seem to introduce instability in the evaluation, which becomes oversensitive to outliers. For example, on the PU1, which contains 1,099 examples, *weighted accuracy* does not work properly when giving λ values of 999, since the misclassification of only one legitimate message leads to score worse than if any email had been filtered (this would give $WAcc = 99.92\%$). Moreover, for 100% precision values, the recall variation from 0% to 100% only affects the measure in 0.08 units.

4.3 Secure Recall

In order to give a clearer picture of the behaviour of classifiers when moving the decision threshold, we include in Figure 3 the precision-recall curves of each classifier. These curves are built giving θ a wide range of values, and computing for each value the recall and precision rates. In these curves, high-precision rates of 100%, 99%, 98% and 95% have been selected so as to obtain the recall rate at these points. Table 6 summarizes these sam-

	$\lambda = 9$				$\lambda = 999$			
	θ	Recall	Prec.	WAcc	θ	Recall	Prec.	WAcc
NB	-	78.77	96.65	96.38	-	46.96	98.80	99.47
Stumps	09.2	89.60	99.08	98.58	16.3	**84.20**	99.51	99.66
TB[1]	10.2	93.55	98.71	98.59	46.9	74.43	**100**	**99.98**
TB[2]	24.1	93.76	98.90	98.76	96.7	76.30	**100**	**99.98**
TB[3]	45.2	91.48	99.32	98.87	126.4	74.01	**100**	**99.98**
TB[4]	19.1	**94.80**	**99.35**	**99.14**	123.1	64.45	**100**	99.97
TB[5]	37.4	93.97	99.12	98.92	178.0	66.53	**100**	99.97

Table 4: Cost–sensitive evaluation results on the PU1 corpus.

ples. All the variants are indistinguishable at level of 95% of precision. However, when moving to higher values of precision ($\geq$ 95%) a significant difference seems to occur between Stumps and the rest of variants using deeper weak rules. This fact proves that increasing the expressiveness of the weak rules can improve the performance when requiring very high precision filters. Unfortunately, no clear conclusions can be drawn about the most appropriate depth. Parenthetically, it can be noted that on the PU1 TreeBoost[4] achieves the best recall rates.

4.4 Experiments with other Feature Sets

The last experiment of this paper explores the performance of boosting classifiers in all the available feature space versions of the PU1 corpus; these differ in the type of features used to represent the messages. For each version of the corpus, namely BARE, LEMM, STOP and LEMM STOP, a TreeBoost[3] classifier has been learned, with up to 2500 rounds of boosting. Since the intention of this experiment is to show if more complex features can potentially perform better, no tuning procedure to get a concrete T value is required. Figure 4 shows the F_1 measure with respect to the number of weak rules used. It can be observed that all classifiers perform very similar (left figure), although slight improvements of 1 point can be obtained with lemmatized features (right figure). These results agree with the ones reported by Androutsopoulos et al. in [1].

5 Conclusions

The experiments presented in this paper show that the AdaBoost learning algorithm is appropiate for the spam email filtering problem. It clearly

	$\lambda = 9$				$\lambda = 999$			
	θ	Recall	Prec.	WAcc	θ	Recall	Prec.	WAcc
NB	-	77.57	99.45	99.43	results not provided			
k-NN	-	81.90	98.80	99.40				
Stack	-	84.80	98.80	99.46				
Stumps	7.0	96.47	99.15	99.76	82.0	52.81	99.61	99.95
TB[1]	26.0	94.80	**99.35**	**99.77**	217.1	50.94	**100**	**99.99**
TB[2]	31.2	95.22	98.92	99.69	209.5	77.55	99.73	99.95
TB[3]	32.5	95.84	99.14	99.75	342.6	74.22	**100**	**99.99**
TB[4]	26.0	95.84	99.14	99.75	337.2	**81.91**	**100**	**99.99**
TB[5]	-9.1	**96.26**	99.14	99.76	451.1	79.83	**100**	**99.99**

Table 5: Cost–sensitive evaluation results on the LingSpam corpus.

PU1

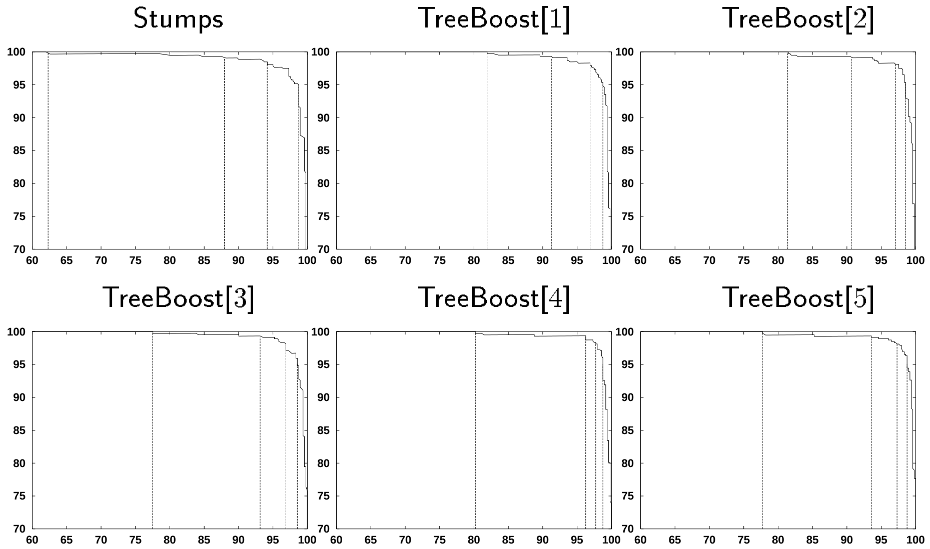
	100%	99%	98%	95%
Stumps	62.37	87.94	94.17	98.75
TreeBoost[1]	**81.91**	91.26	96.88	98.75
TreeBoost[2]	81.49	90.64	97.08	98.54
TreeBoost[3]	77.54	93.13	96.88	98.54
TreeBoost[4]	80.24	**96.25**	**97.71**	**98.75**
TreeBoost[5]	77.75	93.55	97.29	98.75

LingSpam

	100%	99%	98%	95%
Stumps	44.70	97.50	98.38	98.96
TreeBoost[1]	77.75	95.42	98.55	**99.80**
TreeBoost[2]	76.30	94.60	98.55	99.58
TreeBoost[3]	**85.65**	96.25	**98.96**	99.37
TreeBoost[4]	83.99	**96.88**	98.75	99.16
TreeBoost[5]	85.56	96.67	98.54	98.96

Table 6: Recall rate of filtered spam messages with respect to fixed points of high-precision rate

PU1 Corpus



LingSpam Corpus

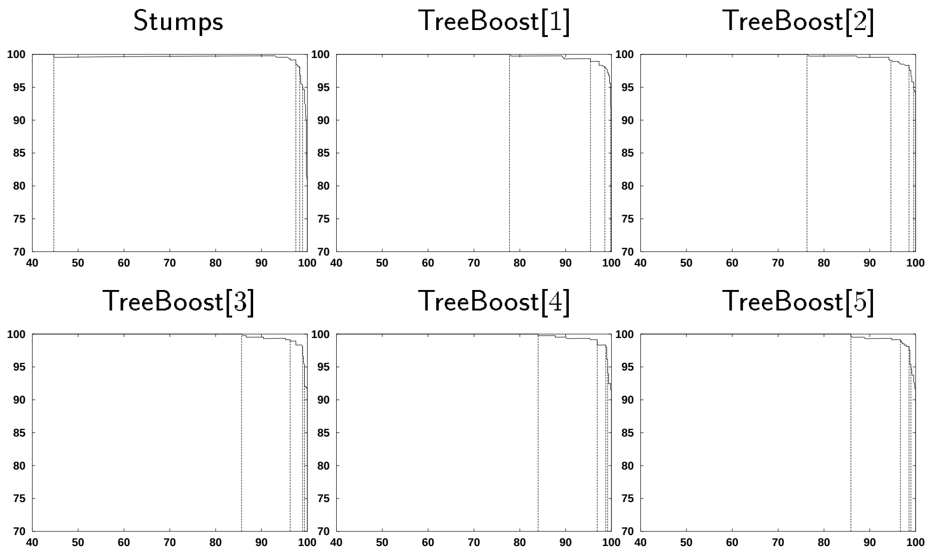


Figure 3: Precision-Recall curves and recall values for the fixed precision rates at 100%, 99%, 98% and 95%. x axis: recall; y axis: precision.

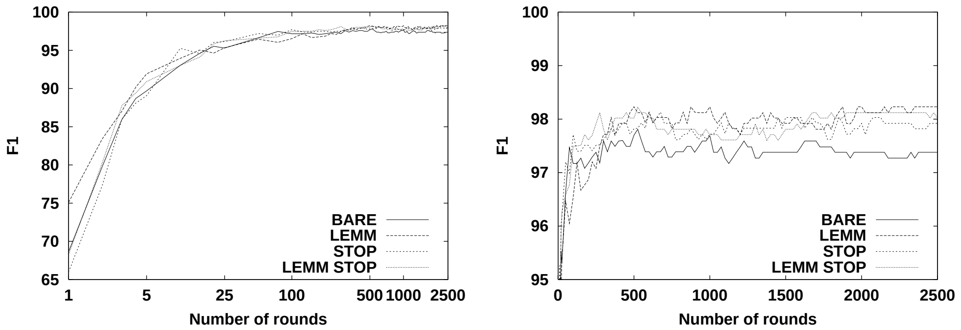


Figure 4: Error rate of TreeBoost[3] classifiers learned with different feature sets on the PU1. The right-hand side plot is a zoom of the left-hand side plot.

outperforms Decision Trees, Naive Bayes and k-NN methods on the two evaluated benchmark corpora: the PU1 corpus and the LingSpam corpus. In these data sets, the method is resistant to overfitting and F_1 rates above 97% are achieved. Procedures for automatically tuning the classifier parameters, such as the number of boosting rounds, are provided.

In scenarios where high-precision classifiers are required, AdaBoost classifiers have been proved to work properly. Experiments have exploited the expressiveness of the weak rules when increasing their depth. It can be concluded that deeper weak rules tend to be more suitable when looking for a very high precision classifier. In this situation, the achieved results on the two corpura are fairly satisfactory.

Two AdaBoost classifiers capabilities have been shown to be useful in final email filtering applications: a) The confidence of the predictions suggests a filter which only classifies the more confident messages, delivering the remaining messages to the final user. b) The classification threshold can be tuned to obtain a very high precision classifier.

6 Future Work

As a future research line, we would like to study the presented techniques in a more difficult corpus. We think that both the PU1 and the LingSpam are too easy, and that the PU1 is also too small. Due to these fatcs, default parameters produce very good results, and the tuning procedures result only in slight improvements. Moreover, some experiments not reported here (which study the effect of the number of rounds, the use of richer feature spaces, etc.) have shown us that the confidence of classifiers depends on several parameters. Using a larger corpus, the effectiveness of the tuning procedures would be more explicit and, hopefully, more significant conclusions about the optimal parameter settings of AdaBoost could be drawn.

In order to overcome these limitations, we are currently working on the development of a new corpus which will contain messages in two languages, English and Spanish, with a realistic proportion of spam messages. Our intention is to make available most of the information in the header and the body of the messages, which will allow to experiment with other feature spaces than the typical 'bag of words' model. For example, we think that information about the components of the sender email address, the number and sequences of some special characters, such as spaces or '$', and n-grams of some words can be relevant for identifying spam messages. Moreover, we want to keep the date and time of messages in the header, which will allow to simulate a filtering system across time.

Another interesting line for future research is the portability of anti-spam filters, that is, to study how a filter learned in a particular data set performs in a different data set. We think that this is a very important issue for general NLP systems which are intended to be final real applications. Since the features provided in the PU1 corpus are encoded and the features in the LingSpam are not, the study of the portability is not possible between these two corpora. Our new future corpus will allow to study the portability of filters against the LingSpam corpus and across time.

Regarding to the learning process, we would like to study the introduction of the misclassification costs inside the AdaBoost learning algorithm. Initial experiments with the method proposed in [16] have not worked properly, although we believe that learning directly classifiers according to some utility settings will perform better than tuning a classifier once learned.

In addition, we would like to experiment with the following two learning algorithms: 1) Alternating Decision Trees [6], for being a boosting-based technique that generalizes the notion of lineraly combined decision trees, and 2) Support Vector Machines, for being the other top-performing method in Text Categorization problems.

Acknowledgments

This research has been partially funded by the EU (IST-1999-12392) and by the Spanish (TIC2000-0335-C03-02, TIC2000-1735-C02-02) and Catalan (1997-SGR-00051) Governments. Xavier Carreras holds a grant by the Department of Universities, Research and Information Society of the Catalan Government.

References

[1] I. Androutsopoulos, J. Koutsias, K. Chandrinos, and C. Spyropoulos. An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–167, 2000.

[2] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. Spyropoulos, and P. Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. In *Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2000)*, 2000.

[3] W. W. Cohen. Learning rules that classify e-mail. In *Proceedings of the 1996 AAAI Spring Symposium in Information Access*, 1996.

[4] L. Cranor and B. LaMacchia. Spam! *Communications of the ACM*, 41(8):74–83, 1998.

[5] H. Drucker, D. Wu, and V.N.Vapnik. Support vector machines for spam categorization. *IEEE Trans. on Neural Networks*, 10(5):1048–1054, 1999.

[6] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the 16th International Conference on Machine Learning, ICML*, 1999.

[7] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

[8] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In C. Nédellec and C. Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398 in Lecture Notes in Computer Science, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[9] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In *Proceedings of the 18th ACM SIGIR Annual Conference*, 1995.

[10] L. Màrquez. *Part–of–Speech Tagging: A Machine–Learning Approach based on Decision Trees.* Phd. Thesis, Dep. Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya, 1999.

[11] S. Robertson and D. A. Hull. The trec-9 filtering track final report. In *The 9-th Text Retrieval Conference (TREC-9)*, 2001.

[12] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization - Papers from the AAAI Workshop*, pages 55–62, 1998.

[13] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C. Spyropoulos, and P. Stamatopoulos. Stacking classifiers for anti-spam filtering of e-mail. In *Proceedings of of the 6th Conference on Empirical Methods in Natural Language Processing (EMNLP 2001)*, pages 44–50, 2000.

[14] R. E. Schapire and Y. Singer. Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning*, 37(3):297–336, 1999.

[15] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[16] R. E. Schapire, Y. Singer, and A. Singhal. Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval, SIGIR '98*, 1998.

[17] S. M. Weiss, C. Apte, F. J. Damerau, D. E. Johnson, F. J. Oles, T. Goetz, and T. Hampp. Maximizing text-mining performance. *IEEE Intelligent Systems*, 14(4):63–69, 1999.

[18] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1999.

[19] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.