An Evaluation Study of Machine Learning Techniques for Identifying Spam

Gaurav R. Bhaya and Harsha V. Madhyastha
Department of Computer Science and Engineering
University of Washington,
Seattle, WA 98195

Abstract

In this work, we investigate the use of two kinds of machine learning techniques - Decision Trees and Naive Bayes - applied to the problem of spam classification. We first consider building a decision tree for this purpose and then, investigate building an ensemble of decision trees using boosting. Decision trees are seen to give fairly good classification accuracy of around 92% and with the use of an ensemble, this accuracy further increases to around 95%. However, overfitting with respect to the training data is also observed in both cases. Our explorations with the Naive Bayes classifier show that it yields extremely high classification accuracy of the order of 99%. Its performance was however found to further improve by using a simpler binomial model assumption and incorporating one of the modifications to Naive Bayes suggested in (Rennie *et al.* 2003).

1 Introduction

One of the most important real-world problems that computer science faces today is putting an end to the ever-increasing amount of spam mail. Being able to correctly classify an email as spam is a challenge that several researchers are looking to address. This problem is nothing but an instance of text classification with two classes - *spam* and *non-spam*. Several machine learning techniques have been applied to solve this problem. In our work, we investigate two such approaches to build a spam classifier - Decision Trees and Naive Bayes. We investigate the working of both these approaches on well-known corpora of labelled spam/non-spam documents.

The first solutions to this problem included *Rule-Based* approaches (Apache 2004; Cunningham *et al.* 2003; Cohen 1996) which were based upon identifying a set of rules to classify an email as *spam* or *non-spam*. Even though this process was automated a simple rule based classification was not sufficient to classify emails because it involved a rigid decision procedure. Some other similar techniques (Orasan & Krishanmurthy 2000) included linguistic analysis of spam emails from a corpus of collected emails.

Some the more popular approaches to Spam filtering included applying popular machine learning algorithms such as Naive Bayes and Decision Trees. Naive Bayes has been successfully applied in many studies on email classification (Lewis & Ringuette 1994; Mitchell 1997; Pantel & Lin 1998). More expressive techniques such as Bayesian classifiers have also been applied to the problem of filtering junk E-mail (Koller & Sahami 1997; Sahami *et al.* 1998). (Rennie *et al.* 2003) proposed various techniques to improve the performance of a Naive Bayes classifier. We implement some techniques proposed in this in our comparative study on spam classification.

Memory based techniques represent another class of successfully applied methods to anti-spam filtering (Sakkis *et al.* 2003). These techniques do not build a model for spam classification but store the training data and evaluate the test instances based on their similarity with respect to training data. While these models adapt well to changes in spammers strategies (Fawcett 2003; Dalvi *et al.* 2004) these techniques are computationally expensive as it requires processing of training data at the time of testing. (Androutsopoulos *et al.* 2000) is a comparative study on memory-based systems and Naive Bayes approaches.

Techniques such as boosting have been shown to work well in the arena of spam classification (Carreras & Marquez 2001). Boosting outperforms the induction of decision trees and is very robust to over-fitting. While this technique is expensive due to the increased complexity of base learners, it allows boosting to obtain higher precisions as compared to decision tree induction.

We first present our experience with use of decision trees in Section 2. Then, in Section 3, we discuss our explorations using an ensemble of decision trees. We follow that with a discussion of our experience with the Naive Bayes classifier in Section 4. Finally, we lay down our conclusions in Section 5.

2 Using Decision Trees

Decision tree induction is one of the simplest and most successful supervised learning algorithm. This section describes our implementation of decision tree induction for the spam classification problem and enlists some techniques that we used to boost the performance of a naive decision tree learner.

At a high level, given the labeled training data classified as *spam* and *not-spam* our decision tree learner learns a binary decision tree which contains one classification attribute

at each node of the tree. The size of the decision tree is influenced by the number of attributes in the data and the size of training data itself. However, the order in which attributes are evaluated in different subtrees are independent. Furthermore, not all attributes may be used to classify the given email as *spam* or *not-spam*. In other words, the depth of leaves may not be the same. However, it is constrained by the number of attributes. We assume that the attributes for classification have been identified apriori using some other techniques and their values are obtained by parsing emails. The attributes used by our classifier correspond to word occurrences, number of capitalized characters in an email, the sender and other attributes known to influence email classification as identified by (Forman *et al.*).

Once the set of attributes have been identified, the next step is to build the decision tree itself. Many techniques have been used to decide the order in which attributes must be considered in a decision tree. The most commonly used technique is based on entropy gain or the amount of information provided by each attribute. Information theory measures information gain in terms of bits i.e., the number of bits of information provided by knowing the value of a given attribute. Mathematically, if the possible answers v_i have probabilities $P(v_i)$, then the information content I of the actual answer is given by:

$$I(P(v_1), P(v_2), .., P(v_n)) = \sum_{i=1}^{n} -P(v_i) log P(v_i) \qquad (1)$$

The information content after classifying the current attribute is the weighted average of the information conveyed by each individual branch of the subtree. The information gain is thus defined as the difference in the information content before and after classification on the given attribute. Based on the entropy values of classification, an attribute that provides the maximum information is chosen for classification.

While the above technique works well if the domain for an attribute is discrete, in the case of the spam classification problem, many of the attributes chosen have continuous values. In such cases, we need to choose an appropriate point in order to split the data into multiple classes. We present some simple heuristics to decide the *point of split* for an attribute with a continuous domain.

- **Mid-point Split:** This is the simplest possible heuristic. It divides the domain of an attribute equally in two halves around the mid point of the domain and hence the name. This heuristic ignores all information provided by the training data and hence is likely to perform poorly.
- **Absolute Error Split:** Next in line to the Mid-point Split, is a heuristic which makes use of the training data in some form to decide a good point to classify data. The Absolute Error based Split divides the data such that the absolute error of classification based on the training data is minimized. For each classified sub-domain the absolute error is defined as the number of elements of the minority class (of that sub-domain) that are present in that sub-domain.
- **Mean of means Split:** The first step involved in classification based on the Mean of means Split is the computation of the mean value of the attribute for each class of data. The mean of individual mean values is then chosen as the value classifying the domain into sub-domains. While this heuristic captures the relative position of all data in a particular class, it does not capture the spread of data elements in a class. It may, thus, fail to find a point of perfect classification (based on the given attribute) if it exists. However, it is a simple and efficient way to compute a *good* point of split.
- **Information Gain:** This heuristic considers all possible positions of split such that the number of elements on each side of the split point is different as compared to the previous point. It chooses the point which provides the maximum information gain. It can be proved that the point of maximum entropy must lie between points which belong to different classes (rather than the same class). This observation reduces the number of positions that need to be considered in order to find the most *informative* split point. This heuristic would find the optimal point of split for a given attribute if it provides complete information.

Evaluation of Decision Trees

In order to evaluate the performance of decision tree learner we implemented a decision tree algorithm in Perl. The program generates a decision tree based on training data in standard XML format. This is then used by other scripts to test its performance on other test data. In order to train and test the learner, we used the data provided by Spambase (Forman *et al.*) which is a repository of over 4000 labeled emails. Using 57 attributes identified by Spambase to the decision tree learner, our evaluations answer the following questions. questions:

1. Can decision trees be used in spam classification?
2. What heuristics (discussed earlier) work well?
3. It is well known that over fitting of data or noisy data affects the performance of decision tree significantly. Does a decision tree learner for spam classification show any signs of over-fitting?
4. How does the learners knowledge increase as more training data is provided to it?

Table 1 shows the performance of four different heuristics discussed earlier on the given data sets. The performance shown in the table is based on 3896 instances of training data and 500 instances of test data. While the naive Mid-point split never misclassified a *non-spam* email as spam, a careful observation reveals that this heuristic based on majority in a class never classified any email as spam. The number of false negatives are very high to consider this heuristic for further study. While the performance of the remaining heuristics is comparable, the Mean of means Split and Information gain heuristics showed less than 1% error on training data and less than 10% error on test data. Mean of means Split classified emails with 91% accuracy with lesser number of false positives than other heuristics. Thus, the table suggests that some heuristics such as Mean of means Split

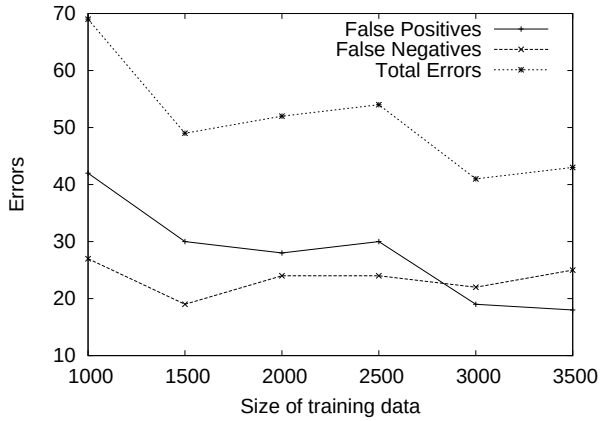

Figure 1: Performance of Mean of mean Split heuristic for different sizes of training data.

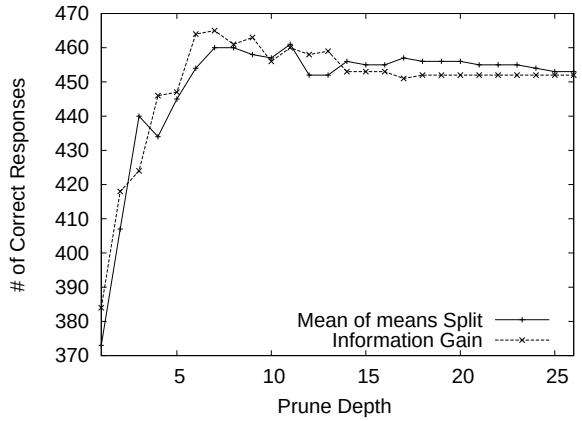

Figure 2: Effect of pruning the decision tree on the performance of Mean of means Split and Information Gain heuristics.

and Information gain work better than others. Furthermore, the table shows that decision tree learning can in fact be applied to the spam classification problem to obtain up to 91% accuracy.

Figure 1 shows the performance of Mean of means Split heuristic as the size of training data is varied. As the size of training data set (sample set) increases, the number of errors on the test data reduces. A significant cause of this reduction appears to be due to the reduction in the number of false positives, while the number of errors due to false negatives is relatively constant.

Figure 2 demonstrates that the decision tree learner overfits the training data, as a result of which the performance of the learner degrades with the increase in the size of the tree. The figure shows that both the heuristics achieve their maximal performance if the tree were pruned at the depth of 7, *i.e.*, all decisions were made on or before the 7th level of the tree. This figure also explains the poorer performance of Information gain heuristic as compared to the Mean of means Split heuristic in Table 1. Pruning the tree at the right level, Information Gain heuristic outperforms the Mean of means heuristic and achieves an accuracy of about 92.8%.

3 Ensemble Learning

Ensemble learning is a common method used to boost the performance of learning algorithms. The idea of ensemble learning methods is to select a whole collection, or ensemble of hypotheses (in this case decision trees) and combine them to make a prediction. Ensemble learning assumes that the errors made by different learners are independent and hence a majority vote is better than any of the individual learners. While this may not completely be true, ensemble learning has been applied successfully in the past.

In the case of the spam classification problem, we use one particular ensemble learning technique called as *boosting*, to improve the quality of our classification. In boosting, each decision tree learner is trained on a subset of data, sampled from a weighted training set. As each learner in the ensemble is trained, the weight of a data item is altered based on the performance of the trained learners on that data item. For example, if all the trained learners classify a given data item incorrectly then its weight is increased while if all learners classify a given data item correctly then its weight is decreased. In our implementation of an ensemble of decision tree learners, the weight of a misclassified data point is doubled. All weights are then normalized.

Evaluation of performance of the Ensemble

In our study we considered an ensemble of up to 9 decision tree learners based on the Mean of means Split and Information gain heuristics. Our experimental setup remains the same as the one described in Section 2. We pose the following questions to evaluate the performance of an ensemble:

1. How many learners need to be present in an ensemble?
2. How does the ensemble perform on training data and on test data?
3. How does over-fitting affect the performance of an ensemble?

Figure 3 shows the performance for various sizes of the ensemble using the Mean of means Split heuristic on the training data itself. Each learner in the ensemble is given randomly selected items from the data set. The number of items given to each learner is captured by the parameter *Sample size*. Clearly as the sample size for each learner is increased the performance of the individual learner improves and hence the performance of the ensemble as a whole. Figure 4 shows a similar result on training data using the Information Gain heuristic. It is worth mentioning that for

Heuristic	Training Data			Test Data		
	Errors	False +	False -	Errors	False +	False -
Mid-point Split	1504 (38.6%)	0	1504	191 (38.2%)	0	191
Absolute Error Split	54 (1.38%)	4	50	61 (12.2%)	25	36
Mean of means Split	23 (0.59%)	1	22	45 (9%)	22	23
Information gain	26 (0.66%)	15	11	48 (9.6%)	25	23

Table 1: Comparison of heuristics for dividing a continuous attribute domain in a decision tree learning algorithm. The results shown here are calculated over 3896 instances of training data and 500 instances of test data.

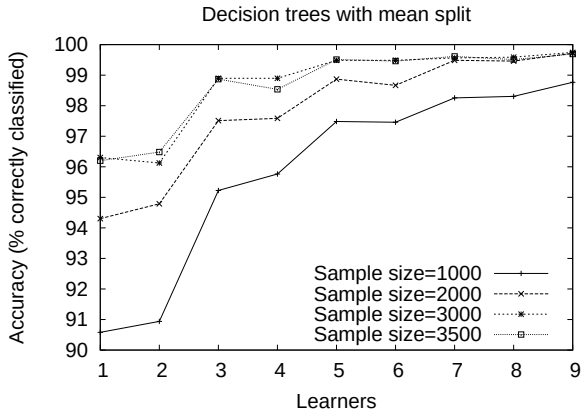

Figure 3: Performance on Mean of means Split ensemble on training data.

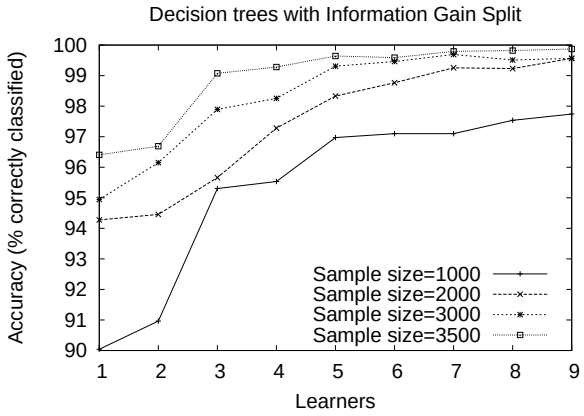

Figure 4: Performance of Information Gain heuristic on training data.

a sample size of 1000 the sudden increase in performance from 2 learners to 3 learners is owing to the increase in the information available to the learners as a whole. This clearly suggests that a sample size of 1000 is insufficient for training a decision tree based learner.

Figures 5 and 6 show the performance of ensemble on test data for Mean of means Split and Information Gain heuristics respectively. While the performance of the ensemble improves with the increase in the number of learners, the figures show that five learners are sufficient to reach the maximum performance of the ensemble. The performance gain due to any additional learners in the ensemble is not significant.

Figure 7 demonstrates the effect of over-fitting on the ensemble. Like a signal learner suffers from over-fitting, an ensemble of decision tree learners also suffers from over-fitting. The performance of a decision tree ensemble is not the best at the maximum depth of the decision tree. However, the figure suggests that the point of optimal performance depends upon the sample size used while training the ensemble. The figure shows that ignoring the effects of over-fitting the ensemble achieves an accuracy of up to 94.8%.

Figures 8 and 9 show the number of votes casted by the ensemble for the expected classification of the emails. The

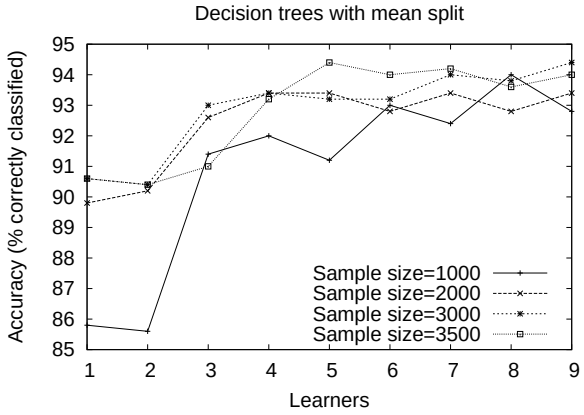

Figure 5: Performance of Mean of means Split heuristic on test data.

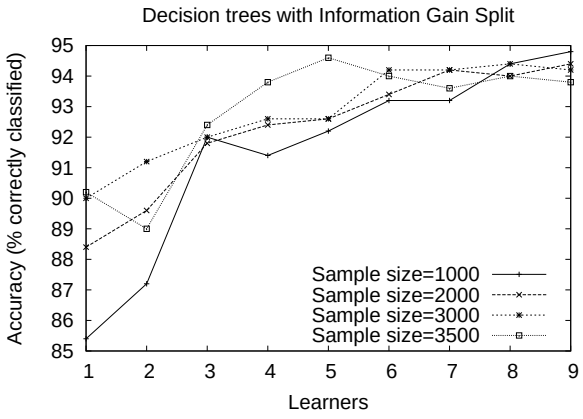

Figure 6: Performance of Information Gain heuristic on test data.

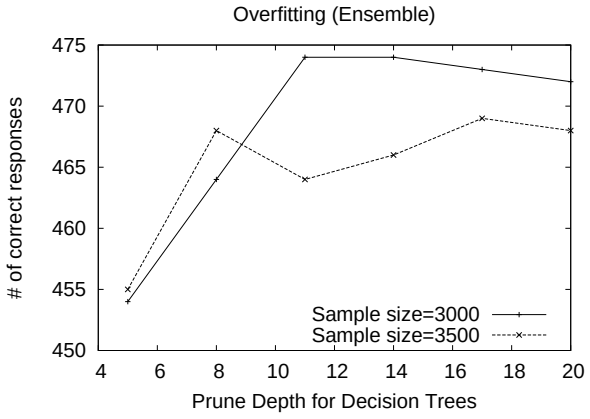

Figure 7: Effect of over-fitting on the ensemble.

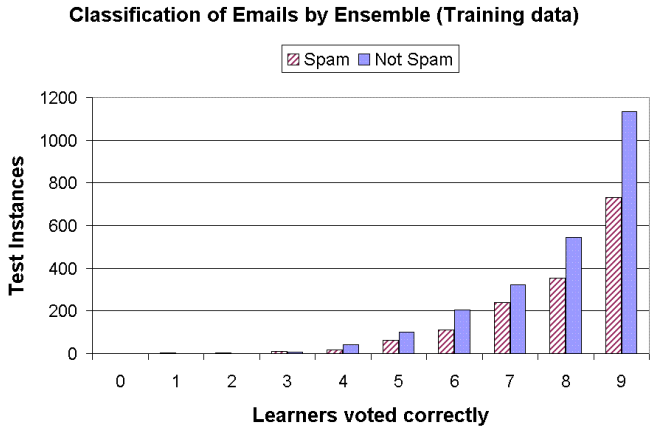

Figure 8: Votes by Ensemble using Information Gain Split on training data.

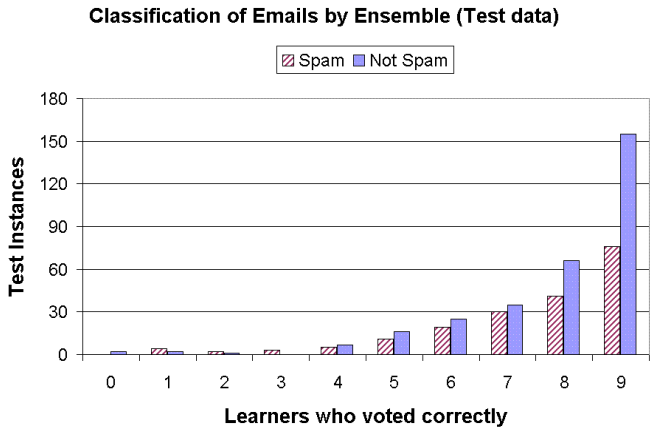

Figure 9: Votes by Ensemble using Information Gain Split on training data.

figures show that both in case of training data and in case of test data, most of the trees in the ensemble vote correctly. Interestingly, the probability that i trees in the ensemble voted correctly is more than (or at least as much as) the probability that only $i-1$ trees voted correctly, both in case of test and training data. As one might expect, these differences are lower in case of test data than in case of training data.

4 Naive Bayes

The second learning approach we used for spam classification is the Naive Bayes technique. Naive Bayes considers every document to be a bag of words and classification is based on the assumption that words assume a multinomial distribution. For classification, there are assumed to be a fixed number of classes, $c \in \{1, 2, \ldots, m\}$ and the parameter vector for every class c is $\theta_c = \{\theta_{c1}, \theta_{c2}, \ldots, \theta_{cn}\}$, where n is the total number of words considered. The like-

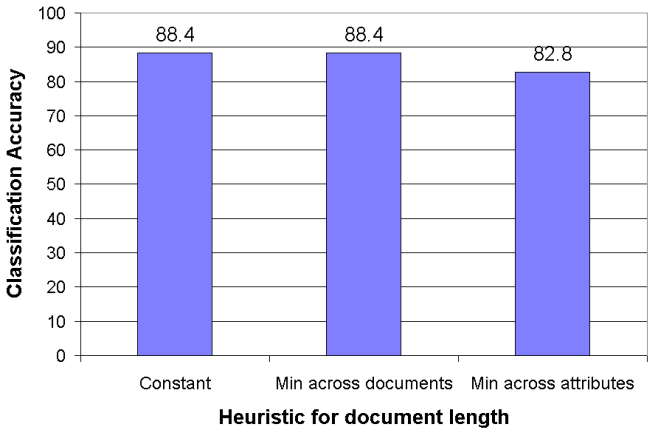

Figure 10: Performance of standard Naive Bayes on the Spambase dataset with different heuristics for estimating document lengths

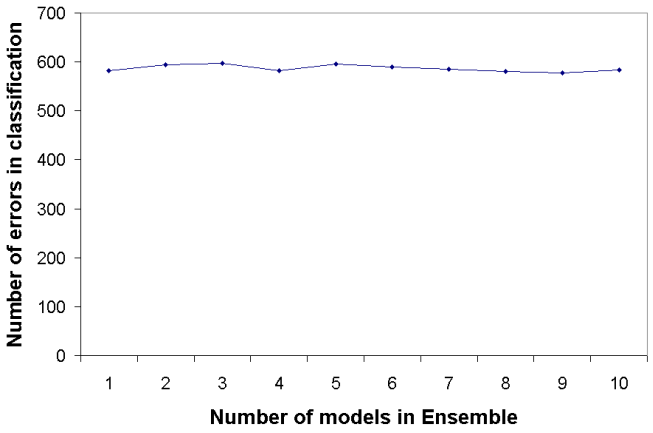

Figure 11: Distribution of number of errors in classification with number of classifiers in the ensemble

lihood of a document d belonging to a class c is given by,

$$p(d|\theta_c) = \frac{(\Sigma_i f_i)!}{\Pi_i f_i!}\Pi_i(\theta_{ci})^{f_i} \quad (2)$$

where f_i is the count of the number of occurrences of word i in document d. Given a prior distribution over the set of classes, $p(\theta_c)$, the classification rule is

$$l(d) = argmax_c[logp(\theta_c) + \Sigma_i f_i log\theta_{ci}] \quad (3)$$

However, before this classification can be applied, the parameters of the model $\theta_{ci}(i \in [1, n])$ and the prior probabilities $p(\theta_c)$ need to be determined. Given a set of documents to train on, these parameters are easily computed as the follows. The estimate for θ_{ci} is simply the number of times word i appears in the documents in class c (N_{ci}), divided by the total number of word occurrences in class c (N_c).

$$\theta_{ci} = \frac{N_{ci} + \alpha_i}{N_c + \alpha} \quad (4)$$

where α_i is a smoothing parameter and α is the sum of all values of α_i. In our work, we consider $\alpha_i = 1$ for all words.

It is clear that the Naive Bayes approach is extremely simple, both in building the model given a set of training documents and in classifying a given document once the model is built. It is due to this simplicity that even though the multinomail distribution assumption is theoretically way off the mark, it is an extremely popular technique chosen to implement. In fact, surprisingly, Naive Bayes has also been known to work pretty well for spam classification inspite of its theoretical frailties. In this work, we study the performance of Naive Bayes in spam classifation and investigate approaches to improve its performance.

Evaluation on Spambase data

We first studied the performance of Naive Bayes on the Spambase database (Forman *et al.*). In this dataset, 48 words have been pre-chosen and for each document, the parameter specified for word i is the fraction of words in that document which word i constitutes. However, for both building the model as well as for classification, Naive Bayes requires the actual count of the number of times each word occurs in every document. Hence, we employed three heuristics for estimating the length of each document.

- **Constant length:** Assume every document is of the same length. Even with this assumption, an actual value for this constant length needs to be chosen. However, in our studies, we found that the performance does not vary much with the choice of this constant length.
- **Min across documents:** For each word, determine the document in which it constitutes the least non-zero fraction of words and assume that in that particular document, this word occurred exactly once. Compute the length of as many documents as possible using this heuristic, and then assign the length of all remaining documents to be the average length of the ones already assigned.
- **Min across attributes:** For each document, determine the word which constitutes the least non-zero fraction of the document (among the chosen words). Assume that this word occurs exactly once in the document and compute the length of the document accordingly. For all those documents, where none of the chosen words occur, assign the length to be the average length of those already assigned.

Using each of these heuristics for estimating the length of every document, we first built the Bayesian model based on the training data set which had approximately 4000 documents. We then considered a test dataset of 500 documents for classification and determined the accuracy with which the Naive Bayes model classifies these documents. Figure 10 shows the accuracy obtained with each of the document length estimation heuristics. Though the Naive Bayes model is seen to predict with almost 90% accuracy, a much

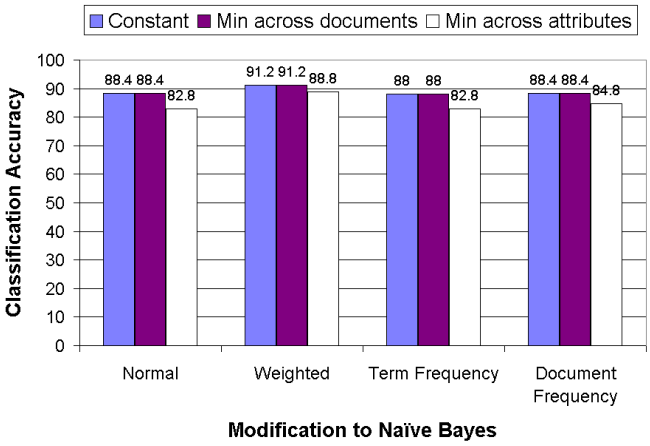

Figure 12: Distribution of classification accuracy on the Spambase dataset with different modifications to Naive Bayes

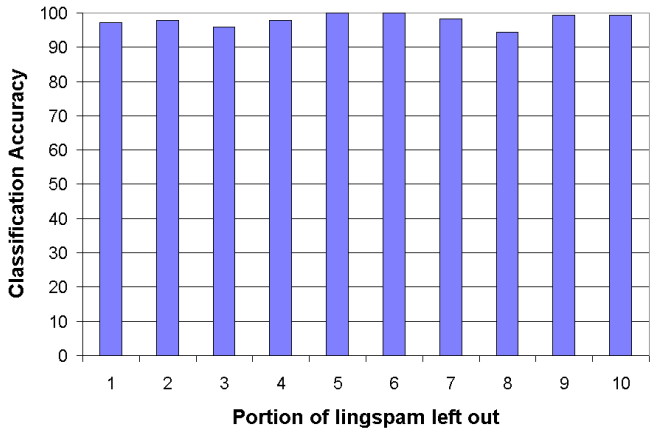

Figure 13: Classification accuracy obtained with the normal Naive Bayes classifier by cross-validation on the Lingspam corpus

better result was expected based on the results obtained in previous studies that used the Naive Bayes classifier.

Drawing inspiration from the success of our use of an ensemble of classifiers in the decision tree domain, we investigated building an ensemble of Naive Bayes classifiers using the same approach as before. In the case of decision trees, we observed that the number of errors in classification reduced with the addition of more classifiers into the ensembler. However, as shown in Figure 11, the number of errors does not drop even after addition of as many as 10 classifiers into the ensemble. So, ensemble construction using boosting did not help in improving the classification accuracy.

We then decided to implement some of the modifications proposed in (Rennie *et al.* 2003), which were intended to help correct the flawed assumptions that the Naive Bayes classifier makes without sacrificing on the simplicity of its implementation. We briefly three of these modifications that we implemented.

- **Weighted:** One of problems that Naive Bayes has is that it does not consider dependencies between words. The modification suggested to take this into account is that, during the classification phase, instead of using $log(\theta_{ci})$ as the weight associated with word i, use $\frac{log(\theta_{ci})}{\Sigma_k |log(\theta_{ck})|}$ instead.
- **Term Frequency:** In most real-world documents, the distribution of word counts is seen to more closely follow a power-law distribution rather than a multinomial distribution. To account for this, a simple transform from statistics is introduced wherein instead of using the count of word i, f_i, we instead use $log(1 + f_i)$.
- **Document Frequency:** It would be useful to "down"-weight words that commonly occur in many documents. For this, the count of word i is transformed from f_i to $f_i \cdot \frac{\Sigma_j 1}{\Sigma_j \delta_{ij}}$, where δ_{ij} is 1 if word i occurs in document j.

We implemented all the 3 modifications to Naive Bayes outlined above and then repeated the study we performed earlier. Figure 12 plots the classification accuracy in each of these cases. It can be observed that none of the modifications helped to significantly increase the classification accuracy. We are still nowhere near the 99% accuracy we hoped to obtain based on empirical studies with Naive Bayes.

Evaluation on Lingspam corpus

In the hope of getting better results with the Naive Bayes classifier, we then decided to investigate its performance on the Lingspam corpus (Trudgian). The Lingspam corpus consists of 10 sets of documents, each of which contains approximately 290 documents. We performed all our experiments on this corpus using the all-but-one cross validation technique, *i.e.*, build the model using all but one of the parts and then test the model on the part that was excluded. Figure 13 shows the classification accuracy produced the the normal Naive Bayes classifier for each of the exempted parts of the Lingspam corpus. We see that now the accuracy shoots up close to the 99% range and in fact, even 100% accuracy is observed in a couple of cases.

Having finally achieved the order of accuracy we were looking for with the Naive Bayes classifier, we wanted to determine if the count of number of times each word occurs in a document is important, or does the mere knowledge of its presence or absence suffice. We built a modified Naive Bayes classifier which discretizes f_i in Equation (3) to 1 or 0 depending on whether word i is present or absent. Figure 14 compares the classification accuracy of this classifier with that obtained with the normal multinomial Naive Bayes classifier. This shows that the binomial model works just as well and in fact, even better than the multinomial model in a few cases.

We then tried out each of the modifications suggested by (Rennie *et al.* 2003) that we outlined previously. Figure 15

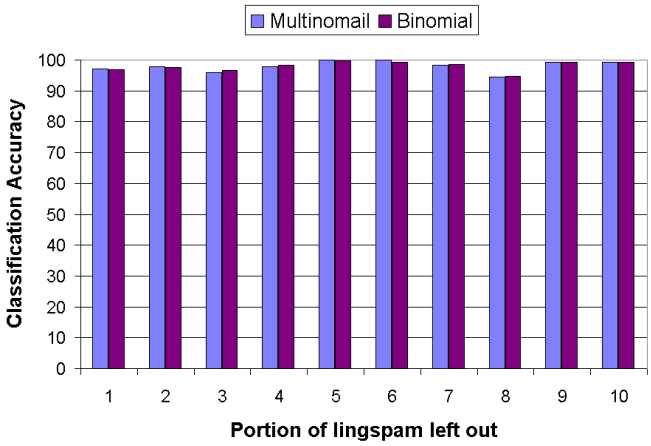

Figure 14: Classification accuracy obtained by cross-validation on the Lingspam corpus using a Naive Bayes classifier which takes into account only presence/absence of each word

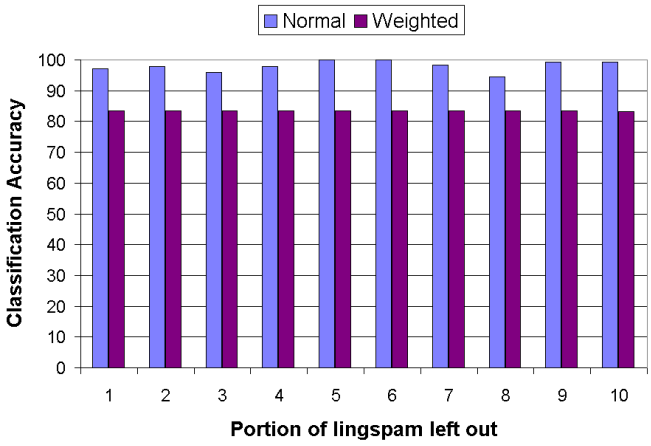

Figure 15: Classification accuracy obtained by cross-validation on the Lingspam corpus using a Naive Bayes classifier with the **Weighted** modification

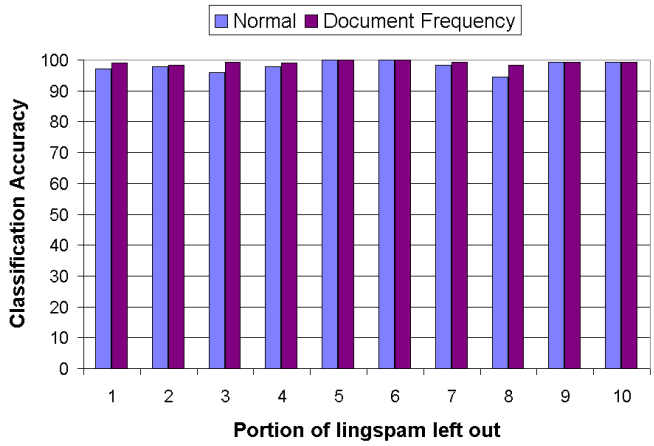

Figure 16: Classification accuracy obtained by cross-validation on the Lingspam corpus using a Naive Bayes classifier with the **Document Frequency** modification

shows that the **Weighted** modification decreases the classification accuracy significantly. On the other hand, Figure 16 shows that the **Document Frequency** modification improves the accuracy in all cases. This indicates that the **Document Frequency** modification is more helpful than the **Weighted** modification. However, we are uncertain whether this inference can be extended to all documents in general or this happens to show up only in the particular corpus of documents we considered.

Figure 17 finally presents a summary of all the variations of Naive Bayes that we investigated. Naive Bayes with the binomial model assumption and with the **Document Frequency** modification seems to be the best algorithm for producing a classifier which minimizes errors in classification.

5 Conclusions

In this paper, we presented the results of our study with two kinds of machine learning techniques - Decision Trees and Naive Bayes - for the purpose of spam classification. Decision trees yielded a fairly high classification accuracy of the order of 92%, but we also observed over-fitting of the model with respect to the training data. We also investigated building of an ensemble of decision tree classifiers, employing the boosting technique, with which the classification accuracy increased to around 95%. However, the problem of over-fitting persisted. Finally, we explored the use of the Naive Bayes classifier, with which we obtained classification accuracies of the order of 99%. We were able to further improve this accuracy by using a simpler binomial model assumption and by incorporating the **Document Frequency** modification suggested in (Rennie *et al.* 2003).

References

[Androutsopoulos *et al.* 2000] Androutsopoulos, I.; Paliouras, G.; Karkaletsis, V.; Sakkis, G.; Spyropoulos, C.; and Stamatopoulos, P. 2000. Learning to filter spam

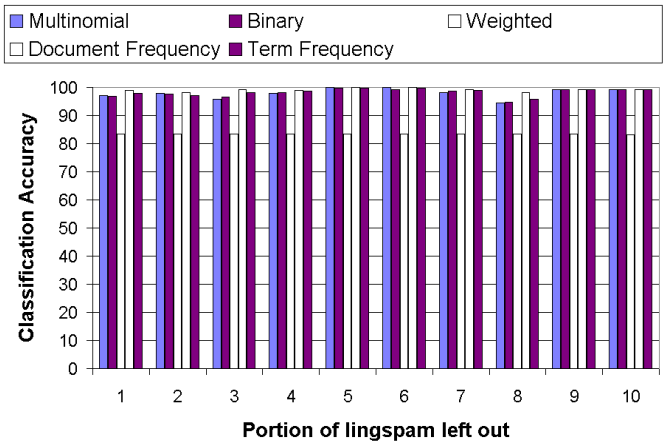

Figure 17: Distribution of classification accuracy on the Lingspam dataset with different modifications to Naive Bayes

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)*.

[Apache 2004] Apache. 2004. Spamassassin. http://spamassassin.apache.org/.

[Carreras & Marquez 2001] Carreras, X., and Marquez, L. 2001. Boosting trees for anti-spam e-mail filtering. In *Proceedings of RANLP2001*, 58–64.

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

[Cunningham *et al*. 2003] Cunningham, P.; Nowlan, N.; Delany, S.; and Haahr, M. 2003. A case-based approach to spam filtering that can track concept drift.

[Dalvi *et al*. 2004] Dalvi, N.; Domingos, P.; Mausam; Sanghai, S.; and Verma, D. 2004. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD*, 99–108.

[Fawcett 2003] Fawcett, T. 2003. In vivo, spam filtering: A challenge problem for kdd. In *SIGKDD Explorations*, volume 5(2), 140–148.

[Forman *et al.*] Forman, G.; Hopkins, M.; Reeber, E.; and Suermondt, J. Spambase database. ftp://ftp.ics.uci.edu/pub/machine-learning-databases/spambase/.

[Koller & Sahami 1997] Koller, D., and Sahami, M. 1997. Hierarchically classifying documents using very few words. In *Machine Learning: Proceedings of the Fourteenth International Conference*, 170–178.

[Lewis & Ringuette 1994] Lewis, D. D., and Ringuette, M. 1994. Comparison of two learning algorithms for text categorization. In *Proceedings of SDAIR*, 81–93.

[Mitchell 1997] Mitchell, T. M. 1997. *Machine Learning*. McGraw Hill.

[Orasan & Krishanmurthy 2000] Orasan, C., and Krishanmurthy, R. 2000. A corpus-based investigation of junk emails. In *Proceedings ACIDCA*.

[Pantel & Lin 1998] Pantel, P., and Lin, D. 1998. Spamcop: (a) spam classification and organization program. In *Learning for Text Categorization: Papers from the 1998 Workshop*. Madison, Wisconsin: AAAI Technical Report WS-98-05.

[Rennie *et al*. 2003] Rennie, J. D.; Shih, L.; Teevan, J.; and Karger, D. R. 2003. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of ICML*, 616–623.

[Sahami *et al*. 1998] Sahami, M.; Dumais, S.; Heckerman, D.; and Horvitz, E. 1998. A bayseian approach to filtering junk e-mail. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*.

[Sakkis *et al*. 2003] Sakkis, G.; Androutsopoulos, I.; Paliouras, G.; Karkaletsis, V.; Spyropoulos, C.; and Stamatopoulos, P. 2003. A memory-based approach to anti-spam filtering for mailing lists. In *Information Retrieval*, volume 6, 49–73.

[Trudgian] Trudgian, D. C. Lingspam corpus. http://www.dcs.ex.ac.uk/corpora/.