Arcing Naive Bayes Classifiers for Spam Filtering: AdaBoost.M1 and Arc-x4

Victor Glazer
Department of Computer Science
University of Toronto
Toronto, ON M5S 3G4
glazer@cs.toronto.edu

Abstract

Naive Bayes classifiers have proven very effective at filtering spam, particularly when stacked or boosted. We use a public spam corpus called Ling-Spam to benchmark the generalization performance of ensembles of Naive Bayes classifiers constructed using AdaBoost.M1, a standard boosting procedure due to Schapire and Freund, and Arc-x4, an adaptive resampling heuristic proposed by Breiman. Both types of ensembles yield significant performance increases over the base classifier, as expected. Moreover, Arc-x4 appears to be competitive with AdaBoost.M1, which is consistent with previous findings by Breiman, Bauer and Kohavi.

1 Introduction

In recent years, spam (unsolicited automated email) has gone from being a nuisance to a serious problem: it wastes untold bandwidth, clogs countless mailboxes and has even prompted some to abandon email altogether. While regulatory approaches may yet prove fruitful, spam filtering currently appears to be the most viable solution. From a Machine Learning standpoint, spam filtering can be viewed as a binary classification problem: a classifier trained on either a public spam corpus or (preferably) on some representative sample of the user's mailbox must categorize incoming email messages as either 'spam' (which we'll denote by '1') or 'legitimate' (denoted by '0'). The use of Naive Bayes classifiers for spam filtering was first suggested (independently) in [9] and [11], and further investigated in [1], [2] and [3]. While the base classifier does surprisingly well, stacking [12] and boosting [8] improve performance significantly. Bagging, on the other hand, is rendered ineffectual by the stability of the Naive Bayes classifier (see, for example, [4]). In this paper we use a public spam corpus called Ling-Spam to benchmark the generalization performance of ensembles of Naive Bayes classifiers constructed using AdaBoost.M1, a standard boosting procedure due to Shapire and Freund described in [7], and Arc-x4, a simple adaptive resampling heuristic proposed by Breiman in [5].

2 Background

2.1 The Ling-Spam corpus

Ling-Spam[1] is a public spam corpus introduced in [1] to evaluate the performance of Naive Bayes spam classifiers. It consists of 2412 legitimate messages randomly selected from Linguist, a moderated linguistics mailing list, and 481 spam messages received by the first author (2893 messages in total). Ling-Spam has a number of disadvantages which limit its usefulness: its size is quite small, spam makes up only about 16.6% of the corpus, all header information has been stripped and the Linguist messages are considerably more homogeneous than a typical user's inbox is likely to be in practice. However, its widespread availability makes it ideal for benchmarking purposes, i.e. for comparing the effectiveness of various spam filtering methods. Another attractive feature of the corpus is that it comes pre-partitioned into 10 roughly equal parts containing approximately the same proportion of spam and legitimate messages, making stratified 10-fold cross validation quick and painless to set up.

2.2 Feature extraction and selection

As noted in [8], Naive Bayes classifiers can easily handle multivalued discrete features such as term frequencies. However, following [1] and [11], we restrict ourselves to binary feature vectors $\vec{X} = (X_1, \ldots, X_M)$ whose i^{th} component X_i indicates whether a particular word occurring in the corpus is present in a given message ($X_i = 1$) or not ($X_i = 0$). As in [1], we rank all available features by their Information Gain (IG) and keep only the M features that yield the highest IG values, where M ranges from 50 to 700 in increments of 50. The Information Gain of feature X was computed as follows:

$$IG(X, C) = \sum_{x \in \{0,1\},\, c \in \{spam,\, legitimate\}} P(x, c) \cdot \log \frac{P(x, c)}{P(x) \cdot P(c)}$$

As shown in [13], such dimensionality reduction (which may be viewed as a form of regularization or capacity control) typically leads to greatly improved generalization performance. Some authors (see, for example, [1] and [9]) have experimented with various lemmatizers and stemmers which reduce words to their base form, as well as stop lists that remove commonly occurring words. These techniques increase preprocessing time and do not appear to lead to significant performance improvements, so we do not use them.

2.3 Naive Bayesian classification

According to Bayes' Theorem, the posterior probability that a message with feature vector $\vec{x}$ belongs to class c is given by

$$P(C = c \mid \vec{X} = \vec{x}) = \frac{P(C = c) \cdot P(\vec{X} = \vec{x} \mid C = c)}{\sum_{k \in \{spam,\, legitimate\}} P(C = k) \cdot P(\vec{X} = \vec{x} \mid C = k)}$$

In Naive Bayesian classification, we make the simplifying assumption that the features $\vec{X}$ are conditionally independent given the class C, i.e. that the class-conditional probabilities *factorize* as

$$P(\vec{X} = \vec{x} \mid C = c) = \prod_{i=1}^{M} P(X_i = x_i \mid C = c)$$

[1] Available for download at http://www.aueb.gr/users/ion/ publications.html

This allows us to rewrite the posterior probability as

$$P(C = c \mid \vec{X} = \vec{x}) = \frac{P(C = c) \cdot \prod_{i=1}^{M} P(X_i = x_i \mid C = c)}{\sum_{k \in \{spam,\, legitimate\}} P(C = k) \cdot \prod_{i=1}^{M} P(X_i = x_i \mid C = k)}$$

The priors $P(C = c)$ and class-conditionals $P(X_i = x_i \mid C = c)$ can now be easily estimated from the training data using relative frequencies with Laplace smoothing. For example, the prior probability that a message is legitimate is computed as

$$P(C = legitimate) = \frac{N_L + 1}{N_L + N_S + 2}$$

where N_L denotes the number of legitimate messages in the corpus, N_S denotes the number of spam messages in the corpus (so that the total number of messages is $N_L + N_S$), and the constants are due to two fictitious messages, one spam and one legitimate, introduced for smoothing purposes. Although the conditional independence assumption isn't particularly realistic, Naive Bayes classifiers have been shown to perform extremely well even when it is violated [6].

In spam classification, false positives, i.e. legitimate messages misclassified as spam, are generally more undesirable than false negatives, i.e. spam messages misclassified as legitimate, since the latter are merely a nuisance (the user need only delete the errant spam messages from his or her mailbox) whereas the former could lead to loss of important information (in case messages classified as spam are deleted immediately rather than saved for subsequent review). We therefore classify a message with feature vector $\vec{x}$ as spam if and only if

$$\frac{P(C = spam \mid \vec{X} = \vec{x})}{P(C = legitimate \mid \vec{X} = \vec{x})} > \lambda$$

where λ, which we call *legitimate message weight*, indicates how much more costly it is to misclassify a legitimate message as spam than to misclassify a spam message as legitimate.

Finally, note that we use $\log$ probabilities throughout for improved numerical stability, which becomes an important issue for Naive Bayes classifiers when the number of features is large [4].

2.4 Performance evaluation measures

The standard performance measure for classification tasks (where by performance we mean generalization performance, i.e. how well a classifier performs on previously unseen testing data) is *Misclassification Error Rate* ($MErr$), defined in this case as

$$MErr = \frac{N_{L \to S} + N_{S \to L}}{N_L + N_S}$$

where N_L and N_S denote the number of legitimate and spam messages to be classified, respectively, $N_{L \to S}$ denotes the number of legitimate messages misclassified as spam and $N_{S \to L}$ denotes the number of spam messages misclassified as legitimate. Two other domain-specific measures introduced in [11] are *Spam Precision* (SP) and *Spam Recall* (SR), defined as

$$SR = \frac{N_{S \to S}}{N_{S \to L} + N_{S \to S}} \qquad SP = \frac{N_{S \to S}}{N_{L \to S} + N_{S \to S}}$$

where $N_{L\to S}$ and $N_{S\to L}$ are defined as before and $N_{S\to S}$ denotes the number of spam messages classified as spam. Intuitively, SR is the fraction of spam messages that were classified as spam, whereas SP is the fraction of the messages that were classified as spam that actually are spam. While ideally both SR and SP should be high, SP is typically the more important of the two, since it relates to the number of false positives whereas SR relates to the number of false negatives. One drawback of using SR and SP to measure performance is that it's not always clear which combination is best. A more serious problem with all three measures ($MErr$, SR and SP) is that they are not cost-sensitive, i.e. they do not take the fact that false positives are generally more undesirable than false negatives into account. In other words, they do not depend on the *legitimate message weight* λ. Androutsopoulos et al. proposed a cost-sensitive performance measure called *Total Cost Ratio* (TCR) in [1].TCR is defined in terms of *Weighted Misclassification Error* ($WMErr$) and *Weighted Baseline Error* ($WBErr$) as follows:

$$WMErr = \frac{\lambda \cdot N_{L\to S} + N_{S\to L}}{\lambda \cdot N_L + N_S} \qquad WBErr = \frac{N_S}{\lambda \cdot N_L + N_S} \qquad TCR = \frac{WBErr}{WMErr}$$

Intuitively, $WMErr$ is just a weighted version of $MErr$, where each legitimate message is treated as λ messages. If a legitimate message is misclassified as spam it counts as λ errors, whereas if it it correctly classified as legitimate it counts as λ successes. $WBErr$ is a "baseline" that measures the weighted error rate in the absence of a filter: while no legitimate messages are misclassified as spam, all spam messages are misclassified as legitimate. The greater $WBErr$ is than $WMErr$ the higher the TCR and the more profitable it is to use the filter. On the other hand, if $TCR < 1$ then $WBErr < WMErr$ and we are better off not using the filter at all.

2.5 Adaptive resampling and combining (arcing)

Breiman originated the concept of arcing in [5] while investigating the differences and similarities between bagging and boosting. An arcing algorithm iteratively constructs an ensemble of K classifiers C_1,...,C_K as follows: initialize the probabilities $\{p(n)\}$ to $p(n) = 1/N$, where $N = |T|$. At the kth step, sample from T with replacement according to $\{p(n)\}$ to get a new training set T^k (note that $T^k \neq T$, in general), and train C_k on T^k. Then update $\{p(n)\}$ so that the training cases that are misclassified more frequently have greater weight. The loop terminates after K iterations, and the classifiers are combined using either majority or weighted voting at test time.

AdaBoost.M1, described in [7], may be implemented as either a resampling or a reweighting procedure, where the latter requires the base classifier to support weighted training cases. The two approaches are largely equivalent, although there is some evidence to support the claim that the reweighting version is superior because it guarantees that all classifiers are trained on all training cases, however weighted (see, for example, [10]). Although its original motivation and theoretical justification come from PAC learning theory, the resampling version of AdaBoost.M1 may be viewed as an arcing algorithm (sometimes called Arc-fs in this context, after the names of its creators, Freund and Schapire): at the kth step, train classifier C_k on the resampled set T^k, then run the original training set T down C_k and let $d(n) = 1$ if the nth training case was classified correctly and $d(n) = 0$ otherwise. Set

$$\epsilon_k = \sum_n p(n) \cdot d(n) \qquad \beta_k = \frac{(1-\epsilon_k)}{\epsilon_k}$$

and update the probabilities $\{p(n)\}$ for the next step as follows:

$$p(n) = \frac{p(n) \cdot \beta_k^{d(n)}}{\sum_n p(n) \cdot \beta_k^{d(n)}}$$

At test time, combine classifiers C_1,..., C_K using weighted voting, with C_k having vote $log(\beta_k)$.

In [5], Breiman argues that the excellent generalization performance of AdaBoost.M1 is due primarily to the adaptive resampling aspect of the algorithm, rather than the fact that the classifiers are combined using weighted voting. He devises an arcing heuristic, dubbed Arc-x4, which combines the classifiers using majority voting and updates the probabilities $\{p(n)\}$ as follows: at the kth step, train classifier C_k on the resampled set T^k, then run the original training set T down C_k and let $m(n)$ be the *total* number of of times the nth training case is misclassified by $C_1, \ldots, C_k$, the classifiers constructed so far. Then update the probabilities $\{p(n)\}$ for the next step as follows:

$$p(n) = \frac{1 + m(n)^4}{\sum_n 1 + m(n)^4}$$

Breiman demonstrates in [5] that Arc-x4 is competitive with AdaBoost.M1 for many real world datasets. In [4], Bauer and Kohavi confirm this to be the case specifically when Naive Bayes is used as the base classifier.

3 Experiments

3.1 Design

We benchmark the generalization performance of AdaBoost.M1 and Arc-x4 on the Ling-Spam corpus, using the *Total Cost Ratio* or TCR as our performance measure. Two different settings of of the *legitimate message weight* λ are used, $\lambda = 1$ and $\lambda = 9$. The former is included mostly for comparison, since in that case $WMErr = MErr$ (refer to Section 2.4 for the definitions of TCR, λ, $WMErr$ and $MErr$). The latter represents a scenario where messages classified as spam are stored in a special folder for subsequent review instead of being immediately deleted. Sahami et al. experimented with $\lambda = 999$ in [11], but that setting was shown to be impractical in [1] due to its instability.

Rather than fixing the number of features M, we repeat all experiments for values of M ranging from 50 to 700 in increments of 50. Although Androutsopoulos et al. have determined the optimal number of features (with respect to maximizing TCR) for $\lambda = 1$ and $\lambda = 9$ in [1], their results concern individual classifiers rather than ensembles.

Following [4] and [10], and in contrast to [5], we restrict ourselves to comparatively small ensembles of size 15 for both AdaBoost.M1 and Arc-x4. Although larger ensembles may yield improved performance, the computational burden they impose is prohibitive in a practical application such as spam filtering.

As in [1], we employ *10-fold stratified cross-validation* to minimize the effects of random variation: the corpus is partitioned into 10 sections of size about 290, each containing approximately 48 spam messages and 242 legitimate messages (the total number of messages is 2893, which does not divide into 10 evenly). We repeat all experiments 10 times, each time holding out a different portion of the corpus, training on the remainder and measuring the TCR on the hold out. The mean TCR is then reported.

Table 1: Best TCR values attained by a single Naive Bayes classifier and by ensembles constructed using the Arc-x4 and AdaBoost.M1 algorithms, over all settings of λ and numbers of features

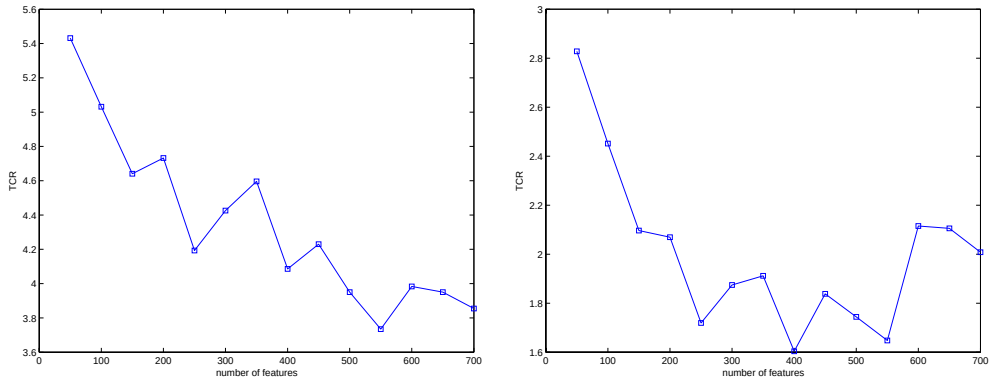
CLASSIFIER TYPE	λ	NUMBER OF FEATURES	TCR
A single Naive Bayes classifier	1	50	5.43
Arc-x4 ensemble	1	450	31.87
AdaBoost.M1 ensemble	1	650	28.12
A single Naive Bayes classifier	9	50	2.83
Arc-x4 ensemble	9	550	8.69
AdaBoost.M1 ensemble	9	550	7.13

3.2 Results

Consulting Table 1, we see that for $\lambda = 1$ ensembles constructed using both AdaBoost.M1 and Arc-x4 yield significant increases in TCR compared to individual Naive Bayes classifiers (486.92% and 417.86%, respectively), with Arc-x4 having an edge over AdaBoost.M1 (see Section 2.4 for the definitions of λ and TCR). The large percentage increases are a little misleading however, since TCR is a cost-sensitive performance measure that is more informative when $\lambda > 1$. In particular, the corresponding changes in SR and SP need not be commensurate (see Section 2.4 for the definitions of SR and SP). Note also that although a single Naive Bayes classifier "peaks early", attaining its best TCR value (5.43) when the number of features is only 50 (as in [1]), the Arc-x4 ensemble attains its best TCR value (31.87) at 450 features, and the AdaBoost.M1 ensemble requires as many as 650 features to attain its best TCR value (28.12).

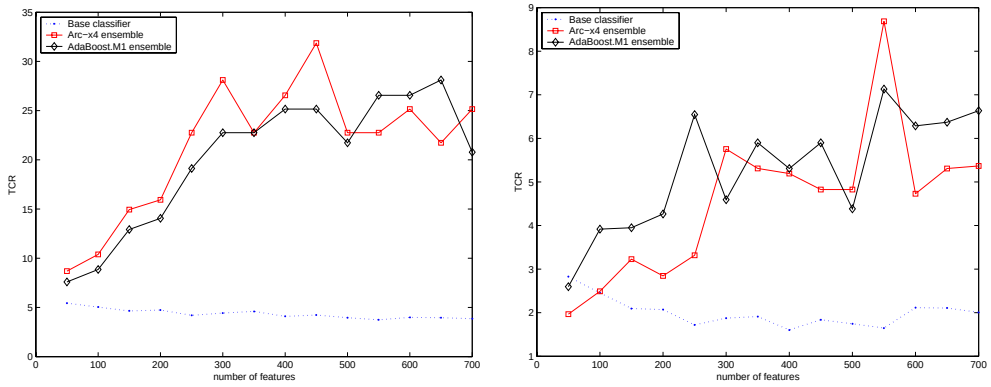
The setting $\lambda = 9$ corresponds more closely to a real world spam-filtering scenario (see Section 3.1). Consulting Table 1 we see that once again both AdaBoost.M1 and Arc-x4 ensembles yield significant performance increases (151.94% and 207.07%, respectively), though not quite as dramatic as for $\lambda = 1$. As before, AdaBoost.M1 trails Arc-x4 somewhat. Although the number of features at which a single classifier attains its best TCR (2.83) is unchanged at 50, this time both Arc-x4 and AdaBoost.M1 ensembles attain their best TCRs (8.69 and 7.13, respectively) at 550 features. This, together with the analogous result for $\lambda = 1$ above suggests that these types of ensembles require a significantly larger number of features to achieve optimal performance than the base classifiers they are composed of.

In terms of the best TCR value attained over all numbers of features, then, Arc-x4 seems to outperform AdaBoost.M1 somewhat. However, consulting Figure 2, which depicts the TCRs attained by the two algorithms as functions of the number of features used, for both settings of λ, we note that the TCR of the AdaBoost.M1 ensemble is a lot "smoother" than the TCR of the Arc-x4 ensemble, which is quite "jagged". Such smoothness presumably indicates that AdaBoost.M1 is a more well-behaved algorithm, in some sense. Moreover, while Arc-x4 yields a higher TCR value overall, it lags behind AdaBoost.M1 for many numbers of features. Figure 1, which depicts the TCRs attained by a single Naive Bayes classifier as a function of the number of features used, for both settings of λ, is included primarily for comparison – due to the difference in scale, the particulars of those curves are difficult to discern in Figure 2.



(a) λ = 1 (b) λ = 9

Figure 1: TCR values attained by a single Naive Bayes classifier as a function of the number of features, used for both $\lambda = 1$ and $\lambda = 9$



(a) λ = 1 (b) λ = 9

Figure 2: TCR values attained by classifier ensembles constructed using Arc-x4 (red) and AdaBoost.M1 (black) as a function of the number of features used, for both $\lambda = 1$ and $\lambda = 9$. For comparison, TCR values attained by a single Naive Bayes classifier (blue) have also been plotted

4 Conclusion

Our results indicate that classifier ensembles constructed using both AdaBoost.M1 and Arc-x4 yield significant increases in generalization performance (as measured by TCR), even if we restrict our attention to comparatively small ensembles containing only about 15 classifiers. The advantage of such small ensembles is that they can be trained fairly quickly and function reasonably fast at test time, which is crucial in a practical application such as spam filtering. Moreover, Arc-x4 appears to be competitive with AdaBoost.M1, outperforming it in a number of cases and attaining a higher TCR value overall. While this might seem surprising in light of Arc-x4's simplicity and lack of formal justification, our results are consistent with previous findings by Breiman [5] and Bauer & Kohavi [4]. Based on the above, the arcing approach to Naive Bayesian spam filtering, whether using AdaBoost.M1 or Arc-x4, seems quite promising and worthy of further investigation.

References

[1] Androutsopoulos, I., Koutsias, J., Chandrinos, K., Paliouras, G., and Spyropoulos, C. (2000) An Evaluation of Naive Bayesian Anti-Spam Filtering. In *Proceedings of the workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning*, pp 9-17.

[2] Androutsopoulos, I., Koutsias, J., Chandrinos, K., and Spyropoulos, C. (2000) 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*, pp. 160-167.

[3] Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C., Stamatopoulos, P. (2000) Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach. In *Proceedings of the workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 1-13.

[4] Bauer, E., and Kohavi, R. (1999) An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, **36**(1-2):105-139.

[5] Breiman, L. (1998) Arcing Classifiers. The Annals of Statistics, **26** (3):801-849.

[6] Domingos, P., and Pazzani, M. (1996) Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In *Proceedings of the 13th International Conference on Machine Learning*, pp 55-63.

[7] Freund, Y., and Schapire, R. (1996) Experiments with a New Boosting Algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pp. 148-156.

[8] Kim, Y.H., Hahn, S.Y., and Zhang, B.Y. (2000) Text Filtering by Boosting Naive Bayes Classifiers. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 168-175.

[9] Pantel, P., and Lin, D. (1998) SpamCop: A Spam Classification & Organization Program. In *Learning for Text Classification – Papers from the AAAI Workshop*, pp. 95-98.

[10] Quinlan, J. (1996) Bagging, Boosting, and C4.5. In *Proceedings of the 13th National Conference On Artificial Intelligence*, pp. 725-730.

[11] Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. (1998) A Bayesian Approach to Filtering Junk E-Mail. In *Learning for Text Classification – Papers from the AAAI Workshop, AAAI Technical Report WS-98-05*, pp. 55-62.

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

[13] Yang, Y., and Pedersen, J. (1997) A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the 14th International Conference on Machine Learning*, pp. 412-420.