Boosting and Microarray Data

Philip M. Long and Vinsensius Berlian Vega
Genome Institute of Singapore

January 2, 2003

Abstract. We have found one reason why AdaBoost tends not to perform well on gene expression data, and identified simple modifications that improve its ability to find accurate class prediction rules. These modifications appear especially to be needed when there is a strong association between expression profiles and class designations. Cross-validation analysis of six microarray datasets with different characteristics suggests that, suitably modified, boosting provides competitive classification accuracy in general.

Sometimes the goal in a microarray analysis is to find a class prediction rule that is not only accurate, but that depends on the level of expression of few genes. Because boosting makes an effort to find genes that are complementary sources of evidence of the correct classification of a tissue sample, it appears especially useful for such gene-efficient class prediction. This appears particularly to be to be true when there is a strong association between expression profiles and class designations, which is often the case for example when comparing tumor and normal samples.

Keywords: Supervised learning, classification, boosting, gene expression data, microarray data,bioinformatics.

1. Introduction

The emerging microarray technology allows scientists to simultaneously measure the level of expression of many genes in a tissue sample. Often, an important component of the analysis of a collection of microarray experiments involves class prediction, in which an algorithm uses the results of these experiments to derive a rule for predicting properties of a tissue sample based on its expression profile. While boosting (Schapire, 1990; Freund, 1995; Freund and Schapire, 1997; Freund and Schapire, 1996) works well on a variety of different types of data, it has appeared that it is not well-suited to expression data (see (Dudoit et al., 2002)). In this paper, we describe one reason why AdaBoost (Freund and Schapire, 1997; Freund and Schapire, 1996), the best-known boosting algorithm, does not perform well on such data, and describe slight modifications that substantially improve its performance.

The idea in boosting algorithms is to combine a number of rough "rules-of-thumb" into a more accurate aggregate class prediction rule (Freund and Schapire, 1999). They work by repeatedly applying a subalgorithm, often called a *base learner*, to a dataset. Before each application, the examples are reweighted to increase the importance of

the examples often classified incorrectly by rules returned by previous applications of the base learner. After a specified number of rounds, all of the class prediction rules returned in the various iterations are aggregated by voting: each rule is assigned a weight, and a tissue is predicted e.g. to be tumor if the total weight of the class prediction rules "voting" for a tumor designation is greater than the total weight of the rules voting for normal.

In this work, as in (Ben-Dor et al., 2000), we consider boosting algorithms in which the base learner outputs the decision stump (Iba and Langley, 1992) with minimum weighted error on the training data. A decision stump bases its prediction on whether a single attribute, in this case the level of expression of a given gene, is above or below a certain threshold: an example would be "predict ALL if $x_{17} \geq 3$, otherwise predict AML", where x_{17} is the level of expression of some gene. Pseudocode for AdaBoost, the standard boosting algorithm, applied in conjunction with decision stumps, is given in Figure 1.

If the base learner outputs a class prediction rule that correctly classifies all of the training data, AdaBoost halts and returns it. For certain expression profile classification problems, like discriminating tumor from normal samples, this happens fairly frequently. Since microarray data often concerns many genes, and relatively few training examples, decision stumps that perform perfectly on the training data often do not perform particularly well on independent test data. On the other hand, many genes in an expression profile provide complementary sources of evidence of the correct classification of a tissue sample. As a result, only paying attention to one such source of evidence can harm performance. These cases account for much of the substandard performance of AdaBoost on microarray data.

We found that adding five artificial random examples, with random class assignments and random expression profiles, to the training data substantially improved the test-set accuracy of AdaBoost: the most natural explanation is that this was primarily due to the effect of reducing the chance that any decision stump perfectly fit the training data. However, it generally tends to increase the training set error, most significantly for stumps that are highly accurate on the training data; this results in the weights with which those stumps vote being brought closer to the weights for the other stumps.

While the performance of the above algorithm that adds noise before applying AdaBoost is interesting, this algorithm is not a satisfactory engineering solution; for one thing, its behavior is too unstable. In this paper, we describe research into the design of algorithms related to AdaBoost that overcome the same problem, but in a more practical way. We have found that a number of small and simple modifications to

AdaBoost lead to algorithms that are practically useful for the analysis of microarray data.

The measures proposed in this paper appear to be needed when there is a strong association between the expression profiles and the class designations. For some classification problems involving microarray data, this is not the case. However, it is often the case when distinguishing for example between tumor and normal samples.

When there is a strong association between expression profiles and class designations, finding an accurate class prediction rule is easy. However, rules that are accurate and also depend on the level of expression of few genes are particularly useful for several reasons (Weston et al., 2000; Guyon et al., 2002; Xing et al., 2001; Li et al., 2002). First, the genes in the classifiers that they output are likely to be particularly important in defining the class designations. It is often reasonable to prioritize these genes for further, more intensive, wet lab research. Also, in the case in which the classes are diseased and normal tissues, the genes found might be hypothesized to play roles analogous to jugular veins in the molecular process driving the diseases; this property is desirable of drug targets. Finally, class prediction rules depending on few genes can result in cheaper diagnostics by enabling antibody-based techniques like ELISA (Parker, 1990) to be employed. Our cross-validation study suggests that, where there is a strong association between expression profiles and class designations, boosting is a particularly valuable tool to identify accurate class prediction rules that depend on the level of expression of few genes.

2. Practical Variants of AdaBoost for Expression Data

In this section, we describe several boosting algorithms customized for expression data. Recall that, for comparison, pseudo-code for AdaBoost is given in Figure 1.

2.1. AdaBoost-VC

We view AdaBoost-VC as the most theoretically principled variant of AdaBoost that we propose.

Our design of AdaBoost-VC is guided by the following commonly adopted point of view (Vapnik and Chervonenkis, 1971; Vapnik, 1982; Valiant, 1984; Vapnik, 1989; Haussler, 1992; Vapnik, 1995; Vapnik, 1998). We assume that a probability distribution over instance/class pairs is used to generate the training data. We further assume that after the algorithm comes up with the classification rule, the instances

on which it must be applied, together with their correct classifications, are also generated according to the same distribution. In the below discussion, it will be useful to consider a collection of random variables, one for each decision stump s, that indicate whether, for a random instance/class pair (x, y), it is the case that $s(x) \neq y$. We will refer to each such random variable as an *error random variable*, or an *error* for short.

Due to the reweighting of the examples, the classification rules returned by different invocations of the base learner tend to have negatively associated errors, say in the sense of (Dubhashi and Ranjan, 1998). Negative association formalizes the idea that a collection of random variables tend to behave differently. Boosting promotes this property in the error random variables by weighting the examples so that examples on which previous decision stumps were incorrect are more important, and thus tend not to be errors for future decision stumps.

When the errors of the decision stumps output by boosting are negatively associated, all else being equal, adding more voters improves the accuracy of the aggregate classifier by reducing the variance of the fraction of voters that correctly classify a random instance, making the correct fraction less likely to dip below 1/2 (this is for a similar reason that adding more independent coin flips reduces the variance of the fraction coming up heads – negative association accentuates this effect (Dubhashi and Ranjan, 1998)). However, when the errors of the individual voting classification rules are unequal, there is a balance to be struck, informally, between the diversity of opinion and its quality. In the case in which the errors are exactly independent, one can work out how optimally to strike this balance (Duda and Hart, 1973): it involves assigning weights to the voters as a function of their accuracy, and taking a weighted vote. To a first approximation, the weighting of the voters computed by AdaBoost might be viewed as akin to this, but taking some account of what dependence there is among the errors.

Intuitively, one would like the errors of the voting classification rules to be negatively associated with respect to the underlying distribution generating the test data. However, some theory (Schapire and Singer, 1999; Kivinen and Warmuth, 1999) suggests that the tendency of the voters in the output of AdaBoost to have negatively associated errors is a byproduct of the more direct effect that the voting classification rules tend to have negatively associated errors with respect to the distribution that assigns equal weight to each of the training examples.

The above viewpoint that AdaBoost approximates finding a set of classification rules with negatively associated errors and then weighting them optimally also suggests that the weights assigned to the voters

should be a function of their accuracy with respect to the underlying distribution. A special case of this is the observation mentioned in the introduction that a voter that is perfect on the training data should not vote with infinitely large weight, as is done in the standard AdaBoost.

In AdaBoost, the weight assigned to a voting classifier, and the reweighting of the examples after it is chosen, is based on the (weighted) error of the voter on the training data. We propose to instead use an estimate of the error with respect to a probability distribution over the entire domain. The probability distribution can be obtained

- starting with the original underlying distribution,
- reweighting every possible instance/class pair according to the number of previously chosen voters that got it wrong in the analogous way as is done by AdaBoost on the training data, and
- normalizing the result so that it is a probability distribution

(i.e., the distribution used in "boosting-by-filtering" (Freund, 1995)).

How to obtain such an estimate? For an individual voter, the weighted error on the training data can be viewed as an estimate of the error according to the reweighted underlying distribution. However, the estimate is biased by the fact that the voter was chosen to minimize this weighted error. Vapnik (1982) proposed to counteract biases like this with a penalty term obtained though a theoretical analysis (Vapnik and Chervonenkis, 1971; Vapnik, 1982). Informally, in this case, this analysis provides bounds on the difference between the observed error rate of the best decision stump and the true error rate with respect to the underlying distribution that hold with high probability for any distribution on the instance/class pairs; Vapnik proposed to adjust the estimate by adding this bound. Kearns, et al (2002) proposed a variant based on a guess of what the result of the tightest possible analysis would be. In our context, if m is the number of examples, n is the number of genes, and ϵ^{emp} is the (weighted) training error, the estimate obtained is

$$\epsilon^{\text{emp}} + \frac{\ln n}{m}\left(1 + \sqrt{1 + \frac{\epsilon^{\text{emp}} m}{\ln n}}\right). \tag{1}$$

(The fact that the estimate is based on a weighted sample weakens the link between their recommendation and this application; if the weight is concentrated in a few examples, the effective number of examples is less than m. Coping with this in a principled way is a potential topic for future research.) The following expression matches theory a little more closely (Vapnik, 1982; Haussler et al., 1994; Talagrand, 1994; Li

et al., 2001)

$$\epsilon^{\text{emp}} + \frac{\ln n}{m}\left(\ln m + \sqrt{1 + \frac{\epsilon^{\text{emp}} m}{\ln n}}\right). \tag{2}$$

(In short, it has been shown that the $\ln m$ term is necessary in the theoretical bounds on how accurate the best decision stump can be.)

Another issue must be confronted: what to do if a classifier returned by the base learner correctly classifies all of the data. Even if (1) or (2) is used, since no errors are made, none of the weights of any of the examples will change, and the base learner will return the same classification rule again the next time it is called, and so on for the remaining number of rounds. We get around this by requiring that a given gene can be used in only one decision stump.

When we began experimentation with an algorithm that used (2) together with only allowing each gene to appear once, it became immediately obvious that the penalty term in (2) was too severe: the estimates were immediately far above 1/2. However, (2) is based on an analysis concerning a worst-case probability distribution. In practice, the "effective" number of genes will be much less. In microarray data, this could be because many genes

- have expression profiles similar to other genes, or
- are completely unassociated with the class label, and therefore present substantially less of a threat to be in decision stumps that fit the data well by chance.

One could imagine estimating the effective number of genes, for example by clustering genes based on their expression profiles and counting the number of clusters with members that correlate significantly with the class label. Instead of incurring the resulting expense in system complexity and computation time, we use the following expression

$$\epsilon^{\text{emp}} + \frac{d}{m}\left(\ln m + \sqrt{1 + \frac{\epsilon^{\text{emp}} m}{d}}\right). \tag{3}$$

with d as an adjustable parameter. In our experiments, we chose d from among $\{0, ..., 3\}$ to minimize five-fold cross-validation error on the training set. In case of a tie, the geometric mean of the values of d attaining the minimum was used.

Pseudo-code for AdaBoost-VC is in Figure 2.

2.2. AdaBoost-NR ("No Repeat")

This algorithm is like AdaBoost, with two changes. First, as in AdaBoost-VC, each gene is constrained to be in at most one decision stump. Second, if a decision stump correctly classifies all of the training data, its weight is set as if its weighted error on the training data was $0.1/m$, where m is the number of samples. This is instead of the infinite weight given to such a stump by AdaBoost. The choice of $0.1/m$ is intended to have the effect, in most cases, of ensuring that the decision stump has the largest weight of those chosen.

We evaluated this algorithm to gain insight into the share of the improvement seen by AdaBoost-VC that could be attributed to using each gene at most once. However, it appears to be a useful algorithm in its own right.

2.3. AdaBoost-PL ("Piecewise Linear")

This algorithm is an instantiation of AdaBoost with "confidence-rated" predictions (Schapire and Singer, 1999). The classes are designated by 1 and -1, and the base classifiers are functions from expression profiles to the continuous interval $[-1, 1]$. When a base classifier h is applied to an expression profile x, the sign of $h(x)$ is interpreted as its class prediction, and the magnitude of $h(x)$ is interpreted as its confidence in that prediction.

The base classifiers used in our implementation of AdaBoost-PL are piecewise-linear generalizations of decision stumps. Note that a decision stump that predicts 1 exactly when $x_i \geq \theta$ can be written as outputting $\text{sign}(x_i - \theta_i)$. This is replaced with $\pi\left(\frac{x_i - \theta_i}{c\sigma_i}\right)$, where

- π is defined by
$$\pi(u) = \begin{cases} 1 & \text{if } u \geq 1 \\ -1 & \text{if } u \leq -1 \\ u & \text{otherwise,} \end{cases}$$
- σ_i is the standard deviation of feature x_i on the training data, and
- c is an adjustable parameter, chosen to minimize five-fold cross-validation error on the training set (the values in $\{0.05, 0.1, 0.2, 0.5, 1.0, 2.0\}$ were tried, and the geometric mean of the values resulting in the minimum error was used).

Similarly, $\text{sign}(-x_i - \theta_i)$ is replaced by $\pi\left(\frac{-x_i - \theta_i}{c\sigma_i}\right)$.

The base classifier h_t of round t is chosen in minimize $\sum_i |h_t(x_i) - y_i| D_t(i)$, where the weights $D_t(i)$ of the examples are updated as in (Schapire and Singer, 1999).

2.4. ARC-X4-RW ("RE-WEIGHT") AND ARC-X4-RW-NR

Since the main problem with AdaBoost on expression data appears to be concentrating too much weight on the predictions of decision stumps that do well on the training data, an anonymous referee asked whether an algorithm like Arc-x4 (Breiman, 1998) might be well-suited to such data. Arc-x4-RW is like boosting, except,

- all base classifiers in the final class prediction rule vote with equal weight, and
- the weight of example i in round t is proportional to $1 + c_{i,t}^4$, where $c_{i,t}$ is the number of base classifiers prior to round t that classified example i incorrectly.

The difference between Arc-x4-RW and Arc-x4 is that, instead of minimizing the weighted training error as in Arc-x4-RW, Arc-x4 resamples from the training set m times with probabilities proportional to the weights, and minimizes the error on the result.

Arc-x4-RW-NR, is like Arc-x4-RW, except with the added constraint that each gene appears in at most one decision stump.

3. Support Vector Machine Algorithms

For comparison, we also evaluated two algorithms that use Support Vector Machines.

3.1. WILCOXON/SVM

This algorithm

- chooses the N genes identified as differentially expressed between the two types of tissues according to the Wilcoxon-Mann-Whitney test with the highest confidence (using the training data), and
- applies SVM with a linear kernel and soft margin with the cost parameter C.

The parameter C is chosen to minimize the five-fold cross-validation error on the training set of the entire inductive process including feature selection – the optimization was done using a simple successive refinement algorithm.

3.2. SVM-RFE

The final algorithm is our implementation of SVM with Recursive Feature Elimination (Guyon et al., 2002). It has a parameter N, the number of genes used.

The data is first rescaled and translated so that each attribute has mean 0 and variance 1 over the training data (the parameters are chosen using the training data, and any test data is rescaled and translated in the same way). Training proceeds in a number of iterations. In each iteration,

- a separating hyperplane is trained using SVM with a linear kernel and the default value of C from SVMlight (Joachims, 1998) (some cross-validation experiments suggested that this performed better than the value $C = 100$ used in (Guyon et al., 2002)),
- the features (in this case genes) are ranked by the absolute magnitude of their corresponding weights in this hyperplane, and
- the bottom ranking half are deleted.

When the last step would reduce the number of genes to less than N, then instead genes are removed from the bottom of the list until N remain.

This is the less computation-intensive of the algorithms proposed by Guyon, et al. It appeared impractical to evaluate the more computation-intensive algorithm in a similar way. It also appeared impractical to choose C using cross-validation on the training set.

4. Experiments

Six datasets were used in our experiments.

- In the well-known ALL-AML dataset (Golub et al., 1999), the task is to determine whether a given gene expression profile belongs to an Acute Lymphoblastic Leukemia (ALL) tissue or an Acute Myeloid Leukemia (AML) tissue. It contains 72 samples (47 ALL, 25 AML), each with expression profiles concerning 7129 genes.
- In a soon-to-be-published dataset on liver cancer (HCC) generated by our colleagues at the Genome Institute of Singapore, there are 76 samples (38 tumor and 38 normal) with expression profiles concerning 9050 genes measured with a cDNA microarray. Ratios against a universal human reference containing a mixture of tissues

types were measured, a log transform was applied, and the data was normalized so that the average log ratio for each array was 0. This data will be made public on the web in the future.

- Another dataset concerns colon cancer (Alon et al., 1999): again, it contains expression profiles for tumor and normal samples.

- The next two datasets analyze expression profiles of breast cancer samples (West et al., 2001) with classes defined by

  - whether the gene responsible for estrogen response is being expressed (ER), and
  - whether the tumor has spread to the lymph nodes (LN).

- The final dataset (Pomeroy et al., 2002) involves predicting whether a patient with a brain tumor survives after treatment.

Aside from the HCC dataset, on which we applied standard preprocessing steps, we used all datasets exactly as we found them.

We evaluated all of the algorithms with two constraints N on the number of genes that they used, 10 and 100. For the boosting-based algorithms, this was achieved by limiting the number of rounds of boosting to N. The use of N in the algorithms used by SVM was described in Section 3.

For each algorithm and each dataset, we performed the following steps 100 times and averaged the results: (a) randomly split into a training set with 2/3 of the examples and a test set with 1/3 of the examples, (b) apply the algorithm on the training set, (c) calculate the error rate on the test set. This is similar to what was done by Dudoit, et al (2002) ; they argued persuasively that this is preferable to more standard techniques like k-fold cross-validation and leave-one-out cross-validation when the goal is to compare the performance of different algorithms, since it reduces the variance of the estimates of the generalization error rates. We subjected all of the algorithms to the same training/test splits, eliminating one source of variance in the estimates of the differences between their average training set errors.

It is worth emphasizing that feature selection was redone using only the training data after each training-test split. Doing cross-validation after feature selection can optimistically bias the resulting error estimates dramatically (Ambroise and McLachlan, 2002; Miller et al., 2002). Also, whenever an algorithm had parameters to set, these were chosen separately for each training-test split, by doing cross-validation on the training set only.

Our results are summarized in Table I.

Table I. Comparison of cross-validation estimates of generalization error percentage of eight algorithms on six microarray datasets.

Algorithm	Gene limit	ALL-AML	HCC	ER	Colon	LN	Brain
Adaboost	10	6.2	7.8	19.9	25.3	40.4	42.3
Adaboost-VC	10	3.9	5.6	18.1	24.4	43.8	41.1
Adaboost-NR	10	3.5	6.0	19.5	25.1	42.7	41.2
Adaboost-PL	10	7.0	7.2	20.6	23.4	36.5	41.9
Arc-x4-RW	10	6.5	8.2	19.8	25.0	39.1	41.4
Arc-x4-RW-NR	10	3.3	5.5	17.8	24.7	42.1	40.7
SVM-RFE	10	13.4	8.6	20.9	19.2	48.4	39.2
Wilcoxon/SVM	10	6.4	6.7	23.2	24.3	35.4	39.3
Adaboost	100	5.2	6.9	16.1	23.4	35.4	38.2
Adaboost-VC	100	2.8	4.8	13.8	22.6	42.8	38.2
Adaboost-NR	100	2.7	4.9	13.2	21.9	40.6	36.5
Adaboost-PL	100	5.0	5.4	17.2	23.2	36.2	38.6
Arc-x4-RW	100	5.4	7.4	16.6	23.7	36.9	38.0
Arc-x4-RW-NR	100	2.6	4.8	12.8	21.6	41.1	36.1
SVM-RFE	100	6.5	6.7	12.6	20.7	48.1	35.7
Wilcoxon/SVM	100	3.3	4.1	17.5	23.6	40.4	37.8

The first observation is that, on the ALL-AML and HCC datasets, where there is a strong association between expression profiles and class designations, AdaBoost-VC, AdaBoost-NR, and Arc-x4-RW-NR all substantially improved on the performance of raw AdaBoost. These algorithms also compare well with the two algorithms using SVM on the ALL-AML and HCC datasets, and to a lesser extent on the ER dataset, especially when only 10 genes are used.

Generally, it appears that as the association between expression profiles and class designations grows weaker, the relative performance of the algorithms using SVM improves.

Arc-x4-RW-NR appears to substantially improve on Arc-x4-RW overall. The additional inductive bias in favor of weighting genes equally appears to be being rewarded. Note that while AdaBoost-VC reduces the weight associated with stumps that perform well on the training data, which has the effect of evening out the weights among the stumps, it also reduces the weights of stumps that perform moderately well on the training data, in some cases reducing them to nearly zero. Thus, overall, the effect of AdaBoost-VC is not necessarily to even out the weights among the voters.

Arc-x4-RW-NR appears to perform the best overall, though its performance on the ALL-AML and HCC datasets is nearly indistiguishable from the performance of AdaBoost-VC and AdaBoost-NR.

The code used in these experiments is available on the web at

`http://giscompute.gis.nus.edu.sg/boost_microarray`

5. Related Work

Despite the fact that our use of a penalty term calls to mind regularization, AdaBoost-VC appears to be quite different from what have been called "regularized boosting algorithms" (Mason et al., 2000; Rätsch et al., 2001). These algorithms are designed based on a theoretical analysis of generalization in ensemble methods (Schapire et al., 1998; Panchenko and Koltchinskii, 2002). This analysis involves two quantities,

- the margin by which a certain fraction of the training data is correctly classified by the aggregate rule, i.e. the minimum difference between the total weight of the correct voters and the total weight of the incorrect voters on any of these examples (after the weights have been normalized to sum to 1), and

- the size of the portion of training data classified with that margin.

As the number of rounds of boosting grows large, the original AdaBoost can be thought of as trying to correctly classify all the examples with a large margin; regularized boosting algorithms are willing to get some of the examples wrong in exchange for a larger margin on the remaining examples, and try to balance these effectively.

When one decision stump can correctly classify all of the data, then an ensemble consisting of only that stump classifies all the data correctly with the largest possible margin, so would be chosen by

these algorithms. Since these cases are the most problematic for AdaBoost, this shows that regularized boosting is not addressing the same limitation as our proposed modifications.

6. Conclusion

In this paper, we have shown how boosting can be adapted to work well on gene expression data, and showed that, appropriately modified, boosting is a useful tool to find small collections of genes that can be used to discriminate between different types of tissues, particularly when there is a strong association between the expression profiles and the class designations.

These encouraging initial results motivate further research into boosting applied to microarray data.

Our modifications appear to be needed due to certain characteristics of expression data: there are typically few examples and many attributes, including a moderately large number that provide approximately independent sources of evidence regarding the class of a tissue. Our modifications are constructed to improve the performance of boosting on data with these characteristics. We have tried applying AdaBoost-VC to two of the standard UC Irvine datasets (`ionosphere` and `promoters`) and found, as we expected, that it performs essentially the same as the unmodified AdaBoost. (Using the same evaluation protocol as described in Section 4, on `ionosphere`, the average test set error of AdaBoost was 10.2%, where AdaBoost-VC obtained 10.8%; on `promoters`, the respective error rates were 10.2% and 10.7%.) However, it is conceivable that algorithms like those proposed here can provide improved performance for other data sources with the key characteristics in common with microarray data. One potential class that we can foresee is text classification problems in which there are few examples. We have not yet looked into this.

It seems likely that combining the ideas of AdaBoost-VC with those behind regularized boosting could lead to a learning algorithm that performs better when the association between expression profiles and class designations is weaker. Some of our initial attempts at this have been unsuccessful, but research is ongoing.

Another potential direction for future work is to estimate the generalization of the decision stumps in a more refined way. One of many possible avenues for this would be to take into account the margin by which the decision stump correctly classifies the training data – previous theoretical work (Vapnik, 1998; Shawe-Taylor et al., 1998; Anthony and Bartlett, 1999; Panchenko and Koltchinskii, 2002) is available to

guide such an estimate. The fact that it appears that spreading the weight evenly among the voters to a greater extent than AdaBoost-VC does appears to be advantageous suggests that the penalty imposed by may be too severe in cases in which the decision stumps do not perform well on the training data.

The penalty used in the estimates of the generalization of the decision stumps in AdaBoost-VC involves the number of examples in the training set. However, if the boosting algorithm concentrates the weight in a few examples, the effective number of examples used for training in a given round might be much less than the original number. Another possible way to refine AdaBoost-VC algorithm would be to take this into account.

Acknowledgements

We especially thank Peter Bartlett, Sanjoy Dasgupta, Wei Fan, Eastwood Leong and Edison Liu for valuable conversations, Shirish Shevade for his comments on an earlier draft of this paper, and anonymous referees for their helpful comments and suggestions.

References

Alon, U., N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, , and A. Levine: 1999, 'Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon cancer tissues probed by oligonucleotide arrays'. *Cell Biology* **96**, 6745–6750.

Ambroise, C. and G. J. McLachlan: 2002, 'Selection bias in gene extraction on the basis of microarray gene-expression data'. *Proc. Natl. Acad. Sci. USA* **99**(10), 6562–6566.

Anthony, M. and P. L. Bartlett: 1999, *Neural Network Learning: Theoretical Foundations*. Cambridge University Press.

Ben-Dor, A., L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini: 2000, 'Tissue Classification with Gene Expression Profiles'. *Journal of Computational Biology* **7**, 559–584.

Breiman, L.: 1998, 'Arcing Classifiers'. *The Annals of Statistics*.

Dubhashi, D. and D. Ranjan: 1998, 'Balls and Bins: A study in negative dependence'. *Random Structures & Algorithms* **13**(2), 99–124.

Duda, R. O. and P. E. Hart: 1973, *Pattern Classification and Scene Analysis*. Wiley.

Dudoit, S., J. Fridlyand, and T. P. Speed: 2002, 'Comparison of discrimination methods for the classification of tumors using gene expression data'. *Journal of the American Statistical Association* **97**(457), 77–87.

Freund, Y.: 1995, 'Boosting a weak learning algorithm by majority'. *Information and Computation* **121**(2), 256–285.

Freund, Y. and R. Schapire: 1996, 'Experiments with a New Boosting Algorithm'. *Proceedings of the Thirteenth International Conference on Machine Learning*.

Freund, Y. and R. Schapire: 1999, 'A short introduction to boosting'. *J. Japan. Soc. for Artif. Intel.* **14**(5), 771–780.

Freund, Y. and R. E. Schapire: 1997, 'A decision-theoretic generalization of on-line learning and an application to boosting'. *Journal of Computer and System Sciences* **55**(1), 119–139.

Golub, T. R., D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander: 1999, 'Molecular classification of cancer: class discovery and class prediction by gene expression monitoring'. *Science* **286**, 531–537.

Guyon, I., J. Weston, S. Barnhill, and V. Vapnik: 2002, 'Gene Selection for Cancer Classification using Support Vector Machines'. *Machine Learning* **46**(1-3), 389–422.

Haussler, D.: 1992, 'Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications'. *Information and Computation* **100**(1), 78–150.

Haussler, D., N. Littlestone, and M. K. Warmuth: 1994, 'Predicting $\{0,1\}$-functions on randomly drawn points'. *Information and Computation* **115**(2), 129–161.

Iba, W. and P. Langley: 1992, 'Induction of One-Level Decision Trees'. *Proc. of the 9th International Workshop on Machine Learning.*

Joachims, T.: 1998, 'Making large-scale support vector machines learning practical'. In: B. S. olkopf, C. Burges, and A. Smola (eds.): *Advances in Kernel Methods: Support vector machines.* pp. 169–184.

Kearns, M., Y. Mansour, A. Y. Ng, and D. Ron: 1997, 'An experimental and theoretical comparison of model selection methods'. *Machine Learning* **27**, 7–50.

Kivinen, J. and M. Warmuth: 1999, 'Boosting as Entropy Projection'. In: *Proc. COLT'99.*

Li, Y., C. Campbell, and M. Tipping: 2002, 'Bayesian automatic relevance determination algorithms for classifying gene expression data'. *Bioinformatics* **18**(10), 1332–1339.

Li, Y., P. M. Long, and A. Srinivasan: 2001, 'Improved bounds on the sample complexity of learning'. *Journal of Computer and System Sciences* **62**(3), 516–527.

Mason, L., P. L. Bartlett, and J. Baxter: 2000, 'Improved Generalization Through Explicit Optimization of Margins'. *Machine Learning* **38**(3), 243–255.

Miller, L. D., P. M. Long, L. Wong, S. Mukherjee, L. M. McShane, and E. T. Liu: 2002, 'Optimal gene expression analysis by microarrays'. *Cancer Cell* **2**(5), 353–361.

Panchenko, D. and V. Koltchinskii: 2002, 'Empirical margin distributions and bounding the generalization error of combined classifiers'. *Annals of Statistics* **30**(1).

Parker, C. W.: 1990, 'Immunoassays'. In: M. P. Deutscher (ed.): *Guide to Protein Purification.* Academic Press.

Pomeroy, S. L., P. Tamayo, M. Gaasenbeek, L. M. Sturla, M. Angelo, M. E. McLaughlin, J. Y. Kim, L. C. Goumnerova, P. M. Black, C. Lau, J. C. Allen, D. Zagzag, J. M. Olson, T. Curran, C. Wetmore, J. A. Biegel, T. Poggio, S. Mukherjee, R. Rifkin, A. Califano, G. Stolovitzky, D. N. Louis, J. P. Mesirov, E. S. Lander, and T. R. Golub: 2002, 'Prediction of central nervous system embryonal tumour outcome based on gene expression'. *Nature* **415**, 436–442.

Rätsch, G., T. Onoda, and K.-R. Müller: 2001, 'Soft Margins for AdaBoost'. *Machine Learning* **42**(3), 287–320. also NeuroCOLT Technical Report NC-TR-1998-021. In press.

Schapire, R. and Y. Singer: 1999, 'Improved boosting algorithms using confidence-rated predictions'. *Machine Learning* **37**(3), 297–336.

Schapire, R. E.: 1990, 'The strength of weak learnability'. *Machine Learning* **5**(2), 197–226.

Schapire, R. E., Y. Freund, P. Bartlett, and W. S. Lee: 1998, 'Boosting the Margin: A new Explanation for the Effectiveness of Voting Methods'. *The Annals of Statistics* **26**(5), 1651–1686.

Shawe-Taylor, J., P. Bartlett, R. Williamson, and M. Anthony: 1998, 'Structural risk minimization over data-dependent hierarchies'. *IEEE Transactions on Information Theory* **44**(5), 1926–1940.

Talagrand, M.: 1994, 'Sharper bounds for Gaussian and empirical processes'. *Annals of Probability* **22**, 28–76.

Valiant, L. G.: 1984, 'A Theory of the Learnable'. *Communications of the ACM* **27**(11), 1134–1142.

Vapnik, V.: 1998, *Statistical Learning Theory*. New York.

Vapnik, V. N.: 1982, *Estimation of Dependencies based on Empirical Data*. Springer Verlag.

Vapnik, V. N.: 1989, 'Inductive principles of the search for empirical dependences (methods based on weak convergence of probability measures)'. *Proceedings of the 1989 Workshop on Computational Learning Theory*.

Vapnik, V. N.: 1995, *The Nature of Statistical Learning Theory*. Springer.

Vapnik, V. N. and A. Y. Chervonenkis: 1971, 'On the uniform convergence of relative frequencies of events to their probabilities'. *Theory of Probability and its Applications* **16**(2), 264–280.

West, M., C. Blanchette, H. Dressman, E. Huang, S. Ishida, R. Spang, H. Zuzan, J. A. O. Jr., J. R. Marks, and J. R. Nevins: 2001, 'Predicting the clinical status of human breast cancer by using gene expression profiles'. *Proc. Natl. Acad. Sci. USA* **98**(20), 11462–11467.

Weston, J., S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik: 2000, 'Feature Selection for SVMs'. In: *NIPS*. pp. 668–674.

Xing, E., M. Jordan, and R. Karp: 2001, 'Feature selection for high-dimensional genomic microarray data'. *Eighteenth International Conference on Machine Learning*.

Given $(x_1, y_1), ..., (x_m, y_m)$ where each $x_i \in \mathbf{R}^n, y_i \in \{-1, 1\}$,

- For each index i of an example, initialize $D_1(i) = 1/m$.
- For each round t from 1 to T:
  - choose a decision stump h_t to minimize the weighted error on the training data with respect to D_t, i.e. to minimize $\sum_{i:h_t(x_i)\neq y_i} D_t(i)$,
  - calculate the error $\epsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$,
  - set the update factor $\beta_t = \epsilon_t/(1-\epsilon_t)$,
  - update the distribution:
    - for each i, set $D'_{t+1}(i) = \begin{cases} \beta_t D_t(i) & \text{if } h_t(x_i) = y_i \\ D_t(i) & \text{otherwise,} \end{cases}$
    - normalize D'_{t+1} to get D_{t+1}, i.e. for each i, set $D_{t+1}(i) = \frac{D'_{t+1}(i)}{\sum_{j=1}^{m} D'_{t+1}(j)}$,
  - set the weight $\alpha_t = \ln \frac{1}{\beta_t}$ with which decision stump t votes (if $\beta_t = 0$, then $\alpha_t = \infty$, and the algorithm can halt).
- Return the final classification rule:

$$h(x) = \begin{cases} 1 & \text{if } \sum_{t:h_t(x)=1} \alpha_t > \sum_{t:h_t(x)=-1} \alpha_t \\ -1 & \text{otherwise.} \end{cases}$$

Figure 1. Pseudo-code for AdaBoost applied with decision stumps (adapted from (Freund and Schapire, 1996))

Given $(x_1, y_1), ..., (x_m, y_m)$ where each $x_i \in \mathbf{R}^n, y_i \in \{-1, 1\}$, and a parameter d,

- For each index i of an example, initialize $D_1(i) = 1/m$, and the set A of available attributes to $\{1, ..., n\}$,
- For each round t from 1 to T:
  - choose a decision stump h_t from among those using attributes in A to minimize the weighted error on the training data with respect to D_t, i.e. to minimize $\sum_{i:h_t(x_i)\neq y_i} D_t(i)$,
  - calculate the weighted empirical error $\epsilon_t^{\text{emp}} = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$,
  - set $\epsilon_t = \epsilon_t^{\text{emp}} + \frac{d}{m}\left(\ln m + \sqrt{1 + \frac{\epsilon_t^{\text{emp}} m}{d}}\right)$,
  - set the update factor $\beta_t = \epsilon_t/(1 - \epsilon_t)$,
  - update the distribution:
    - for each i, set $D'_{t+1}(i) = \begin{cases} \beta_t D_t(i) & \text{if } h_t(x_i) = y_i \\ D_t(i) & \text{otherwise,} \end{cases}$
    - normalize D'_{t+1} to get D_{t+1}, i.e. for each i, set $D_{t+1}(i) = \frac{D'_{t+1}(i)}{\sum_{j=1}^{m} D'_{t+1}(j)}$,
  - set the weight $\alpha_t = \ln \frac{1}{\beta_t}$ with which decision stump t votes,
  - remove the attribute used in h_t from the set A of available attributes.
- Return the final classification rule:

$$h(x) = \begin{cases} 1 & \text{if } \sum_{t:h_t(x)=1} \alpha_t > \sum_{t:h_t(x)=-1} \alpha_t \\ -1 & \text{otherwise.} \end{cases}$$

Figure 2. Pseudo-code for AdaBoost-VC