Machine Learning, 61, 71–103, 2005

DOI: 10.1007/s10994-005-1123-6

The Synergy Between PAV and AdaBoost

W. JOHN WILBUR wilbur@ncbi.nlm.nih.gov
LANA YEGANOVA
WON KIM
National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bethesda, MD, U.S.A.

Editor: Robert Schapire
Published online: 08 June 2005

Abstract. Schapire and Singer's improved version of AdaBoost for handling weak hypotheses with confidence rated predictions represents an important advance in the theory and practice of boosting. Its success results from a more efficient use of information in weak hypotheses during updating. Instead of simple binary voting a weak hypothesis is allowed to vote for or against a classification with a variable strength or confidence. The Pool Adjacent Violators (PAV) algorithm is a method for converting a score into a probability. We show how PAV may be applied to a weak hypothesis to yield a new weak hypothesis which is in a sense an ideal confidence rated prediction and that this leads to an optimal updating for AdaBoost. The result is a new algorithm which we term PAV-AdaBoost. We give several examples illustrating problems for which this new algorithm provides advantages in performance.

Keywords: boosting, isotonic regression, convergence, document classification, *k* nearest neighbors

1. Introduction

Boosting is a technique for improving learning by repeatedly refocusing a learner on those parts of the training data where it has not yet proved effective. Perhaps the most successful boosting algorithm is AdaBoost first introduced by Freund and Schapire (1997). Since its publication AdaBoost has been the focus of considerable study and in addition a large number of related boosting algorithms have been put forward for consideration (Aslam, 2000; Bennett, Demiriz, & Shawe-Taylor, 2000; Collins, Schapire, & Singer, 2002; Duffy & Helmbold, 1999, 2000; Friedman, Hastie, & Tibshirani, 2000; Mason, Bartlett, & Baxter, 2000; Meir, El-Yaniv, & Ben-David, 2000; Nock & Sebban, 2001; Ratsch, Mika, & Warmuth, 2001; Ratsch, Onoda, & Muller, 2001; Schapire & Singer, 1999)(see also http://www.boosting.org). Our interest in this paper is the form of AdaBoost using confidence-rated predictions introduced by Schapire and Singer (1999). For clarity we will refer to this form of AdaBoost as CR(confidence-rated)-AdaBoost as opposed to the binary form of AdaBoost originally introduced by Freund and Schapire (1997).

In the CR-AdaBoost setting a weak hypothesis not only votes for the category of a data point by the sign of the number it assigns to that data point, but it provides additional information in the magnitude of the number it assigns. CR-AdaBoost combines the confidence-rated weak hypotheses that are generated into a linear combination with constant coefficients to produce its final hypothesis. While this approach works well in many cases,

it is not necessarily ideal. A particular weak hypothesis may erroneously invert the order of importance attached to two data points or if the order is not inverted it may still assign an importance that is not consistent between two data points. As a potential cure for such problems we here consider the Pool Adjacent Violators algorithm (Ayer et al., 1954; Hardle, 1991). This algorithm makes use of the ordering supplied by a weak hypothesis and produces the isotonic regression which in a sense is the ideal confidence-rated prediction consistent with the ordering imposed by the weak hypothesis. We show how this may be converted into a resulting weak hypothesis which produces optimal improvements at each step of what we term PAV-AdaBoost. Our algorithm is a type of real AdaBoost, as Friedman, Hastie, and Tibshirani (2000) use the general term, but their realization involves trees, while our development is in the direction of linearly ordered structures. Isotonic regression (PAV) could potentially play a role in other approaches to boosting provided the weak hypotheses generated have the appropriate monotonicity property. We chose CR-Adaboost because the confidence of a prediction $h(x)$ is assumed to go up with its magnitude and this is exactly the condition where isotonic regression is appropriate.

In the most general setting PAV-AdaBoost may be applied exactly parallel to CR-AdaBoost. However, there are situations where one is limited to a certain fixed number of weak hypotheses. To deal with this case we propose a modified form of PAV-AdaBoost which repeatedly cycles through this finite set of weak hypotheses until no more improvement can be obtained. The result is a type of back-fitting algorithm, as studied by Buja, Hastie, and Tibshirani (1989) but the fitting is nonlinear and convergence does not follow from their results. For CR-AdaBoost with a fixed finite set of weak hypotheses, Collins, Schapire, and Singer (2002) have recently given a proof of convergence to the optimum under mild conditions. While our algorithm does not satisfy their hypotheses we are able to prove that it also converges to the optimum solution under general conditions.

The first part of this paper (Sections 2 and 3) deals with theoretical issues pertinent to defining the algorithm. Section 2 consists of an introduction to the PAV algorithm together with pseudocode for its efficient implementation. Section 3 recalls CR-AdaBoost (Schapire & Singer, 1999), describes how PAV is used with CR-AdaBoost to yield the PAV-AdaBoost algorithm, and proves several optimality results for PAV-AdaBoost. The second part of the paper (Sections 4 and 5) deals with the learning capacity and generalizability of the algorithm. Section 4 gives three different examples where PAV-AdaBoost is able to learn on training data more successfully than confidence-rated AdaBoost. In its full generality PAV-AdaBoost has infinite VC dimension as shown in an appendix. In Section 5 we show that an approximate form of PAV-AdaBoost has finite VC dimension and give error bounds for this form of PAV-AdaBoost. All the results in this paper are produced using the approximate form of PAV-AdaBoost. The final sections present examples of successful applications of PAV-AdaBoost. In Section 6 we consider a real data set and a simulated problem involving two multidimensional normal distributions. Both of these examples involve a fixed small set of weak hypotheses. For these examples we find PAV-AdaBoost outperforms CR-AdaBoost and compares favorably with several other classifiers tested on the same data. The example in Section 7 deals with a real life document classification problem. CR-AdaBoost applied to decision tree weak hypotheses has given perhaps the best overall document classification performance to date (Apte, Damerau, & Weiss, 1998;

Carreras & Marquez, 2001). The method by which CR-AdaBoost (Schapire & Singer, 1999) deals with decision tree weak learners is already optimized so that PAV-AdaBoost in this case does not lead to anything new. However, there are examples of weak learners unrelated to decision trees and we consider two types for which PAV-AdaBoost is quite effective, but CR-AdaBoost performs poorly. Section 8 is a discussion of the practical limitations of PAV-AdaBoost.

2. Pool adjacent violators

Suppose the probability $p(s)$ of an event is a monotonically non-decreasing function of some ordered parameter s. If we have data recording the presence or absence of that event at different values of s, the data will in general be probabilistic and noisy and the monotonic nature of $p(s)$ may not be apparent. The Pool Adjacent Violators (PAV) Algorithm (Ayer et al., 1954; Hardle, 1991) is a simple and efficient algorithm to derive from the data that monotonically non-decreasing estimate of $p(s)$ which assigns maximal likelihood to the data. The algorithm is conveniently applied to data of the form $\{(s_i, w_i, p_i)\}_{i=1}^N$ where s_i is a parameter value, w_i is the weight or importance of the data at i, and p_i is the probability estimate of the data associated with i. We may assume that no value s_i is repeated in the set. If a value were repeated we simply replace all the points that have the given value s_i with a single new point that has the same value s_i, for which p_i is the weighted average of the contributing points, and for which w_i is the sum of the weights for the contributing points. We may also assume the data is in order so that $i < j \Rightarrow s_i < s_j$. Then the s_i, which serve only to order the data, may be ignored in our discussion.

We seek a set of probabilities $\{q_i\}_{i=1}^N$ such that

$$i < j \Rightarrow q_i \leq q_j \tag{1}$$

and

$$\prod_{i=1}^{N} q_i^{w_i p_i} (1 - q_i)^{w_i (1 - p_i)} \tag{2}$$

is a maximum. In other words we seek a maximal likelihood solution. If the sequence $\{q_i\}_{i=1}^N$ is a constant value q, then by elementary calculus it is easily shown that q is the weighted average of the p's and this is the unique maximal likelihood solution. If the q's are not constant, they in any case allow us to partition the set N into K nonempty subsets defined by a set of intervals $\{(r(j), t(j)\}_{j=1}^K$ on each of which q takes a different constant value. Then we must have

$$q_i = \frac{\sum_{k=r(j)}^{t(j)} w_k p_k}{\sum_{k=r(j)}^{t(j)} w_k}, \qquad r(j) \leq i \leq t(j) \tag{3}$$

by the same elementary methods already mentioned. We will refer to these intervals as the *pools* and ask what properties characterize them. First, for any k, $r(j) \leq k \leq t(j)$

$$\frac{\sum_{i=r(j)}^{k} w_i p_i}{\sum_{i=r(j)}^{k} w_i} \geq \frac{\sum_{i=r(j)}^{t(j)} w_i p_i}{\sum_{i=r(j)}^{t(j)} w_i} \geq \frac{\sum_{i=k}^{t(j)} w_i p_i}{\sum_{i=k}^{t(j)} w_i}. \tag{4}$$

This must be true because if one of the $\geq$'s in inequality (4) could be replaced by $<$ for some k, then the pool could be broken at the corresponding k into two pools for which the solutions as given by Eq. (3) would give a better fit (a higher likelihood) in Eq. (2). Let us call intervals that satisfy the condition (4) *irreducible*. The pools then are irreducible. They are not only irreducible, they are maximal among irreducible intervals. This follows from the fact that each successive pool from left to right has a greater weighted average and the irreducible nature of each pool.

It turns out the pools are uniquely characterized by being maximal irreducible intervals. This follows from the observation that the union of two overlapping irreducible intervals is again an irreducible interval. It remains to show that pools exist and how to find them. For this purpose suppose I and J are irreducible intervals. If the last integer of I plus one equals the first integer of J and if

$$\frac{\sum_{i\in I} w_i p_i}{\sum_{i\in I} w_i} \geq \frac{\sum_{i\in J} w_i p_i}{\sum_{i\in J} w_i} \tag{5}$$

we say that I and J are *adjacent violators*. Given such a pair of violators it is evident from condition (4), for each interval, and from condition (5) that the pair can be pooled (a union of the two) to obtain a larger irreducible set. This is the origin of the PAV algorithm. One begins with all sets consisting of single integers (degenerate intervals) from 0 to $N-1$. All such sets are irreducible by default. Adjacent violators are then pooled until this process of pooling is no longer applicable. The end result is a partition of the space into pools which provides the solution (3) to the order constrained maximization problems (1) and (2).

As an illustration of irreducible sets and pooling to obtain them consider the set of points:

$$w_0 = 1, \ldots, w_5 = 1 \quad \text{and} \quad p_0 = \frac{1}{4},\ p_1 = \frac{1}{3},\ p_2 = \frac{1}{5},\ p_3 = \frac{1}{4},\ p_4 = 1,\ p_5 = \frac{1}{2}$$

One pass through this set of points identifies two pairs $\{p_1, p_2\}$ and $\{p_4, p_5\}$ as adjacent violators. When these are pooled they result in single points of weight 2 with the values $\frac{4}{15}$ and $\frac{3}{4}$, respectively. A second pass through the resulting set of four points shows that $\{p_1, p_2\}$ and p_3 are now adjacent violators. These may be pooled to produce the pool $\{p_1, p_2, p_3\}$and the value $\frac{47}{180}$. Examination shows that there are no further violations so that the irreducible pools are $\{p_0\}$, $\{p_1, p_2, p_3\}$, and$\{p_4, p_5\}$.

In order to obtain an efficient implementation of PAV one must use some care in how the pooling of violators is done. A scheme we use is to keep two arrays of pointers of length N. An array of forward pointers, F, keeps at the beginning of each irreducible interval the position of the beginning of the next interval (or N). Likewise an array of backward

pointers, B, keeps at the beginning of each interval the position of the beginning of the previous interval (or -1). Pseudocode for the algorithm follows.

2.1. *PAV algorithm*

#Initialization
A. Set $F[k] \leftarrow k+1$ and $B[k] \leftarrow k-1,\ 0 \le k < N$.
B. Set up two (FIFO) queues, Q_1 and Q_2, and put all the numbers, $0, \ldots, N-1$, in order into Q_1 while Q_2 remains empty.
C. Define a marker variable m and a *flag*.
#Processing
While (Q_1 nonempty){
  Set $m \leftarrow 0$.
  While (Q_1 nonempty){
    $k \leftarrow$ dequeue Q_1.
    If($m \le k$){
      Set *flag* $\leftarrow 0$.
      While ($F[k] \neq N$ and intervals at k and $F[k]$ are adjacent violators){
        Pool intervals beginning at k and $F[k]$.
        Set $u \leftarrow F[F[k]]$.
        Redefine $F[k] \leftarrow u$.
        If ($u < N$) redefine $B[u] \leftarrow k$.
        Update $m \leftarrow u$.
        Update *flag* $\leftarrow 1$.
      }
    }
    If (*flag* $= 1$ and $B[k] \ge 0$) enqueue $B[k]$ in Q_2.
  }
  Interchange Q_1 and Q_2.
}

This is an efficient algorithm because after the initial filling of Q_1 an element is added to Q_2 only if a pooling took place. Thus in a single invocation of the algorithm a total of no more than N elements can ever be added to Q_2 and this in turn limits the number of tests for violators to at most $2N$. We may conclude that both the space and time complexity of PAV are O(N). See also Pardalos and Xue (1999).

PAV produces as its output the pools described above and these define the solution as in Eq. (3). It will be convenient to define $p(s_i) = q_i$, $i = 1, \ldots, N$ and we will generally refer to the function p.

A simple example of the type of data to which the algorithm may be applied is a Bernoulli process for which the probability of success is a monotonic function of some parameter. Data generated by such a process could take the form of tuples such as $(s, 3, 2/3)$. Here the parameter value is s and at this value there were 3 trials and 2 of these were a success. PAV applied to such data will give us a prediction for the probability of success as a function

of the parameter. The PAV algorithm may be applied to any totally ordered data set and assigns a probability estimate to those parameter values that actually occur in the data set. For our applications it will be important to interpolate and extrapolate from these assigned values to obtain an estimated probability for any parameter value. We do this in perhaps the simplest possible way based on the values of p just defined on the points $\{s_i\}_{i=1}^N$.

2.2. *Interpolated PAV*

Given the function p defined on the set $\{s_i\}_{i=1}^N$, assume that the points come from a totally ordered space, S, and are listed in increasing order. Then for any $s \in S$ define $p(s)$ by

Case 1. If $s = s_i$, set $p(s) = p(s_i)$.
Case 2. If $s < s_1$, set $p(s) = p(s_1)$.
Case 3. If $s_i < s < s_{i+1}$, set $p(s) = \frac{p(s_i)+p(s_{i+1})}{2}$.
Case 4. If $s_N < s$, set $p(s) = p(s_N)$.

While one could desire a smoother interpolation, 2.2 is general and proves adequate for our purposes. In cases where data is sparse and the parameter s belongs to the real numbers, one might replace Case 3 by a linear interpolation. However, there is no optimal solution to the interpolation problem without further assumptions. The definition given has proved adequate for our applications and the examples given in this paper.

PAV allows us to learn a functional relationship between a parameter and the probability of an event. The interpolation 2.2 simply lets us generalize and apply this learning to new parameter values not seen in the training data.

3. AdaBoost

The AdaBoost algorithm was first put forward by Freund and Schapire (1997) but improvements were made in Schapire and Singer (1999). We first review important aspects of CR-AdaBoost as detailed in Schapire and Singer (1999), and then describe how PAV is used with AdaBoost. The AdaBoost algorithm as given in Schapire and Singer (1999) follows.

3.1. *AdaBoost algorithm*

Given training data $\{(x_i, y_i)\}_{i=1}^m$ with $x_i \in X$ and $y_i \in \{1, -1\}$, initialize $D_1(i) = 1/m$. Then for $t = 1, \ldots, T$:

(i) Train weak learner using distribution D_t.
(ii) Get weak hypothesis $h_t : X \to \mathbb{R}$.
(iii) Choose $\alpha_t \in \mathbb{R}$.
(iv) Update:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_t h_t(x_i))}{Z_t}$$

where

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i)). \tag{6}$$

Output the final hypothesis:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right). \tag{7}$$

Schapire and Singer establish an error bound on the final hypothesis.

Theorem 1. *The training error of H satisfies*:

$$\frac{1}{m}\left|\{i|H(x_i) \neq y_i\}\right| \leq \prod_{t=1}^{T} Z_t. \tag{8}$$

From inequality (8) it is evident that one would like to minimize the value of Z_t at each stage of AdaBoost. Schapire and Singer suggest two basic approaches to accomplish this. The first approach is a straightforward minimization of the right side of Eq. (6) by setting the derivative with respect to α_t to zero and solving for α_t. They observe that in all interesting cases there is a unique solution which corresponds to a minimum and may be found by numerical methods. We shall refer to this as the *linear* form of AdaBoost because it yields the sum

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{9}$$

which is a linear combination of the weak hypotheses.

The second approach to optimization presented in Schapire and Singer (1999) is applicable to the special case when the weak learner produces a partition of the training space into mutually disjoint nonempty sets $\{X_j\}_{j=1}^{N}$. Here α is absorbed into h, h is assumed to be a constant c_j on each set X_j, and one asks how to define the c_j in order to minimize Z in Eq. (6). Let us define a measure μ on all subsets of X by

$$\mu(A) = \sum_{x_i \in A} D(i).$$

Then we may set

$$\begin{aligned} W_+^j &= \mu(\{x_i \mid x_i \in X_j \wedge y_i = 1\}) \\ W_-^j &= \mu(\{x_i \mid x_i \in X_j \wedge y_i = -1\}) \end{aligned} \tag{10}$$

The solution (Schapire & Singer, 1999) is then given by

$$c_j = \frac{1}{2} \ln \left(\frac{W_+^j}{W_-^j} \right) \tag{11}$$

and we may write

$$Z = 2 \sum_{j=1}^{N} \sqrt{W_+^j W_-^j} = 2 \sum_{j=1}^{N} \sqrt{u_j (1 - u_j)} \mu(X_j) \tag{12}$$

provided we define

$$u_j = \frac{W_+^j}{\mu(X_j)}$$

3.2. PAV-AdaBoost algorithm

The AdaBoost Algorithm 3.1 is modified from step (iii) on. The weak hypothesis coming from step (ii) is assumed to represent a parameter so that $s_i = h_t(x_i)$. Each data point (x_i, y_i) may thus be transformed to a tuple

$$(s_i, D_t(i), (1 + y_i)/2). \tag{13}$$

PAV is applied to this set of tuples to produce the function $p_t(s)$. With the use of $p_t(s)$ we can define another function

$$k_t(s) = \frac{1}{2} \ln \left(\frac{p_t(s)}{1 - p_t(s)} \right). \tag{14}$$

Clearly this function is monotonic non-decreasing because $p_t(s)$ is. There is one difference, namely, $k_t(s)$ may assume infinite values. Through the formula (11) this leads to the replacement of Eq. (9) by

$$f(x) = \sum_{t=1}^{T} k_t(h_t(x)) \tag{15}$$

and Eq. (12) becomes

$$Z_t = 2\int \sqrt{p_t(s)(1-p_t(s))}\, d\mu(s) = 2\int \sigma_t(s)\, d\mu(s) \tag{16}$$

where $\mu(s)$ is the measure induced on $h_t[X]$ by μ through h_t. Based on Eq. (16) and the inequality

$$2\sqrt{p(1-p)} = \sqrt{1-(1-2p)^2} \le 1 - 2(p-1/2)^2$$

we may obtain the bound

$$Z_t \le 1 - 2\int (p_t(s) - 1/2)^2 d\mu(s).$$

Further we have the following theorem.

Theorem 2. *Among all monotonic non-decreasing functions defined on $h_t[X]$, k_t is that function which when composed with h_t produces the weak hypothesis with minimum Z_t.*

Proof: By its construction $p_t(s)$ partitions the set $h_t[X]$ into a family of subsets $\{S_j\}_{j=1}^m$ on each of which p_t, and hence k_t, is a constant. For each j set $X_j = h_t^{-1}[S_j]$. Then referring to the defining relations (10) we see that the function $W_+^j e^{-c} + W_-^j e^c$ achieves its unique minimum when c is the constant value of k_t on S_j. Now let g be a monotonic non-decreasing function that when composed with h_t produces the minimum Z_t. We first claim that g is constant on each set S_j. Suppose that g is not constant on some S_j and let the minimal value of g be c' and maximal value of g be c'' on S_j. Let S' and S'' be those subsets of S_j that map to c' and c'', respectively, under g. Then in turn set $X' = h_t^{-1}[S']$ and $X'' = h_t^{-1}[S'']$ and define W'_+ , etc., in the obvious way as in Eq. (10). Then we have the inequality

$$\frac{W'_+}{W'_-} \ge \frac{W_+^j}{W_-^j} \ge \frac{W''_+}{W''_-} \tag{17}$$

which follows from the constancy of p_t on S_j (see inequality (4) above). Now suppose c' is less than the constant value of k_t on S_j. Then because the function $W'_+ e^{-c} + W'_- e^c$ is convex with a strictly positive second derivative it decreases in value in moving c from c' to the value of k_t as a consequence of inequality (17). Thus we can decrease the value of Z_t arising from g by increasing c' to the next greater value of g or to k_t whichever comes first. This contradicts the optimality of g. We conclude that c' cannot be less then the value of k_t on S_j. Then c'' must be greater than the constant value of k_t on S_j. But the dual argument to that just given shows that this is also impossible. We are forced to the conclusion that g must be a constant on S_j. Now if g were not equal to k_t on S_j, we could decrease the value of Z_t arising from g by changing its value on S_j to that of k_t, which we know is optimal.

This may cause g to no longer be monotonic non-decreasing. However if we make this change for all S_j the result is k_t, which is monotonic non-decreasing. The only conclusion then is that k_t is optimal. □

There are practical issues that arise in applying PAV-AdaBoost. First, we find it useful to bound p_t away from 0 and 1 by ε. We generally take ε to be the reciprocal of the size of the training set. This is the value used in Schapire and Singer (1999) for "smoothing" and has the same purpose. If we assume that $0 < \varepsilon < 1/2$ and define $\lambda = \ln[(1-\varepsilon)/\varepsilon]$, then this approach is equivalent to truncating k_t at size λ. Define

$$k_{\lambda t}(s) = \begin{cases} k_t(s), & |k_t(s)| \leq \lambda \\ \lambda, & \lambda < k_t(s) \\ -\lambda, & -\lambda > k_t(s). \end{cases} \tag{18}$$

Based on the proof of Theorem 2 we have the following corollary.

Corollary 1. *Among all monotonic non-decreasing functions defined on $h_t[X]$ and bounded by λ, $k_{\lambda t}$ is that function which when composed with h_t produces the weak hypothesis with minimum Z_t.*

Proof: Theorem 2 tells us that among monotonic non-decreasing functions, $k_t$is that function that minimizes Z_t. Let W be a pool of points from $h_t[X]$ on which k_t takes the constant value $\beta(>\lambda)$. We already know that β minimizes the function

$$F(c) = W_+ e^{-c} + W_- e^{c}. \tag{19}$$

Because this function has a strictly positive second derivative and achieves its minimum at β the value that minimizes it within $[-\lambda, \lambda]$ is λ. A similar argument applies if $\beta < -\lambda$. Thus $k_{\lambda t}$ must be that function which is monotonic non-decreasing and constant on each pool and minimizes Z_t. The only question that remains is whether there could be a monotonic non-decreasing function bounded by $[-\lambda, \lambda]$ but not constant on the pools and which yields a smaller Z_t. That this cannot be so follows by the same argument used in the proof of the Theorem. □

In some applications we apply the weak learner just as outlined above. This seems appropriate when there is an unlimited supply of weak hypotheses h_t produced by the weak learner. Just as for standard AdaBoost we try to choose a near optimal weak hypothesis on each round. In other applications we are limited to a fixed set $\{h_j\}_{j=1}^n$ of weak hypotheses. In this case we seek a set $\{k_{\lambda j}\}_{j=1}^n$ of monotonic non-decreasing functions bounded by λ that minimize the expression

$$\sum_{i=1}^{m} \exp\left[-y_i \sum_{j=1}^{n} k_{\lambda j}(h_j(x_i))\right]. \tag{20}$$

Because each $k_{\lambda j}$ in expression (20) acts on only the finite set $h_j[X]$ and the result is bounded in the interval $[-\lambda, \lambda]$, compactness arguments show that at least one set $\{k_{\lambda j}\}_{j=1}^n$ must exist which achieves the minimum in (20). We propose the following algorithm to achieve this minimum.

3.3. *PAV-AdaBoost.finite algorithm*

Assume a fixed set of weak hypotheses $\{h_j\}_{j=1}^n$ and a positive real value λ. For any set $\{k_{\lambda j}\}_{j=1}^n$ of monotonic non-decreasing functions bounded by λ and where each $k_{\lambda j}$ is defined on $h_j[X]$ define

$$q_i = \exp\left[-y_i \sum_{j=1}^{n} k_{\lambda j}(h_j(x_i))\right]. \tag{21}$$

To begin let each $k_{\lambda j}$ be equal to the 0 function on $h_j[X]$. Then one round of updating will consist of updating each $k_{\lambda j}$, $1 \le j \le n$, in turn (not in parallel) according to the following steps:

(i) Define

$$D_i = \frac{q_i \exp[y_i k_{\lambda j}(h_j(x_i))]}{Z_0} \tag{22}$$

where

$$Z_0 = \sum_{i=1}^{m} q_i \exp[y_i k_{\lambda j}(h_j(x_i))]. \tag{23}$$

(ii) Apply the PAV algorithm as described under 3.2 and Corollary 1 to produce the optimal $k_{\lambda j}$ which minimizes $\sum_{i=1}^m D_i \exp[-y_i k_{\lambda j}(h_j(x_i))]$. Replace the old by the new $k_{\lambda j}$.

At each update step in this algorithm either $k_{\lambda j}$ is unchanged or it changes and the sum in (20) decreases. Repetition of the algorithm converges to the minimum for expression (20).

Theorem 3. *Let $\{k_{\lambda j}^t\}_{j=1}^n$ denote the functions produced on the t<u>th</u> round of Algorithm* 3.3. *Let $\{k'_{\lambda j}\}_{j=1}^n$ denote any set of functions that produces the minimum in expression* (20). *Then we have*

$$\lim_{t\to\infty} \sum_{j=1}^{n} k_{\lambda j}^t \circ h_j = \sum_{j=1}^{n} k'_{\lambda j} \circ h_j \tag{24}$$

pointwise on X.

Proof: Clearly for $\{k'_{\lambda j}\}_{j=1}^{n}$ the update procedure of Algorithm 3.3 can produce no change. Let $\{q_i^t\}_{i=1}^{m}$ denote the result of (21) applied to $\{k_{\lambda j}^t\}_{j=1}^{n}$, where t denotes the result of the tth round. By compactness we may choose a subsequence $\{t(r)\}$ for which each $\{k_{\lambda j}^{t(r)}\}_{r=1}^{\infty}$ converges pointwise to a function $k_{\lambda j}$. For this set of functions $\{k_{\lambda j}\}_{j=1}^{n}$ let $\{q_i^*\}$ denote the values defined by Eq. (21). We claim that the set $\{k_{\lambda j}\}_{j=1}^{n}$ is invariant under the update procedure of the algorithm. If not suppose that the update procedure applied to this set of functions produces an improvement when $k_{\lambda j'}$ is updated and decreases the sum (20) by an $\varepsilon > 0$. Then because the update procedure is a continuous process, there exists an integer $M > 0$ such that when $r > M$ the update procedure applied in the $t(r) + 1$th round to $k_{\lambda j'}^{t(r)}$ must produce an improvement in the sum (20) by at least $\varepsilon/2$. However, the sums $\sum_{i=1}^{m} q_i^{t(r)}$ converge downward to $\sum_{i=1}^{m} q_i^*$ so that we may choose an $N > M$ for which $r > N$ implies

$$\sum_{i=1}^{m} q_i^{t(r)} - \sum_{i=1}^{m} q_i^* < \frac{\varepsilon}{4} \tag{25}$$

It follows then that

$$\sum_{i=1}^{m} q_i^{t(r)+1} < \sum_{i=1}^{m} q_i^*. \tag{26}$$

But this latter relation contradicts the fact that the sum $\sum_{i=1}^{m} q_i^*$ must be a lower bound for any such sum produced by the algorithm because of the monotonically decreasing nature of these sums in successive steps in the algorithm. We thus conclude that $\{k_{\lambda j}\}_{j=1}^{n}$ must indeed be invariant under the algorithm's update procedure.

The next step is to show that an invariant set under the algorithm's update procedure defines a unique function on X. For any $\alpha, 0 \leq \alpha \leq 1$, consider the set of functions $\{\alpha k'_{\lambda i} + (1-\alpha)k_{\lambda i}\}_{i=1}^{m}$. This is again a set of monotonic functions to be considered in taking the minimum of expression (20). Define

$$F(\alpha) = \sum_{i=1}^{m} \exp\left[-y_i \sum_{j=1}^{n} \{\alpha k'_{\lambda j}(h_j(x_i)) + (1-\alpha)k_{\lambda j}(h_j(x_i))\}\right]. \tag{27}$$

By assumption $F(1)$ is the true minimum of (20) while $F(0) = \sum_{i=1}^{m} q_i^*$. Let us suppose that $F(1) < F(0)$. Because e^{-x} is a convex function it follows that F is also and we have

$$F(\alpha) \leq \alpha F(1) + (1-\alpha)F(0). \tag{28}$$

From this relation it follows that

$$F'(0) \leq F(1) - F(0) < 0. \tag{29}$$

Now define

$$G(\alpha_1, \alpha_2, \ldots, \alpha_n) = \sum_{i=1}^{m} \exp\left[-y_i \sum_{j=1}^{n} \{\alpha_j k'_{\lambda j}(h_j(x_i)) + (1-\alpha_j) k_{\lambda j}(h_j(x_i))\}\right] \quad (30)$$

and note that with $\alpha_j = \alpha,\ 1 \le j \le n$, we have

$$F(\alpha) = G(\alpha_1, \alpha_2, \ldots, \alpha_n). \quad (31)$$

It then follows that

$$F'(0) = \sum_{j=1}^{n} \frac{\partial G}{\partial \alpha_j}(0, 0, \ldots, 0). \quad (32)$$

Then some one of the partials of G in Eq. (32) must be negative, say the one corresponding to j'. But then the nature of the derivative ensures that for some $\alpha > 0$

$$G(0, 0, \ldots, \alpha_{j'}(\alpha), \ldots, 0) < G(0, 0, \ldots, 0). \quad (33)$$

This, however, contradicts the invariance of the set $\{k_{\lambda j}\}_{j=1}^{n}$ under the update procedure applied at j'. We are forced to the conclusion that $F(0) = F(1)$. Now suppose there were an i with

$$\sum_{j=1}^{n} k'_{\lambda j}(h_j(x_i)) \neq \sum_{j=1}^{n} k_{\lambda j}(h_j(x_i)). \quad (34)$$

Then because e^{-x} is strictly convex we would have for any α, $0 < \alpha < 1$, the relation

$$F(\alpha) < \alpha F(1) + (1-\alpha) F(0) = F(1). \quad (35)$$

But this contradicts the minimality of $F(1)$. We conclude that inequality (34) does not occur. At this point we have shown that

$$\lim_{r\to\infty} \sum_{j=1}^{n} k_{\lambda j}^{t(r)} \circ h_j = \sum_{j=1}^{n} k'_{\lambda j} \circ h_j \quad (36)$$

where the convergence is pointwise on X. It now follows easily that convergence holds as in Eq. (36) not just for the subsequence $\{t(r)\}_{r=1}^{\infty}$, but also for the original sequence. If not we could choose a subsequence of the original sequence that for some particular x_i converged to a value other than $\sum_{j=1}^{n} k'_{\lambda j}(h_j(x_i))$. By compactness we could then choose a subsequence of this subsequence to take the place of $\{t(r)\}_{r=1}^{\infty}$ in the above arguments. The end result must then be a contradiction because inequality (34) must be true in this case. This establishes the convergence stated in Eq. (24). □

For evaluation purposes we generally work with $f(x)$ as defined in Eq. (15) rather than $H(x) = sign(f(x))$. In this way we not only use confidence-rated predictions as input, but produce a confidence-rated prediction as output. We believe this allows for a more sensitive analysis of performance. The function $f(x)$ is used as a ranking function. This allows measures such as recall and precision to be applied. If there are a total of A items with label $+1$ and if among the r top scoring items there are a that have label $+1$, then the *precision* and *recall* at rank r are calculated as

$$P_r = \frac{a}{r} \quad \text{and} \quad R_r = \frac{a}{A} \tag{37}$$

Both precision and recall vary between 0 and 1, but not all values are taken for any given data set. This has led to the concept of an interpolated precision corresponding to a prescribed recall level. If R is any number between 0 and 1 the *interpolated* precision at recall level R is defined

$$IP_R = \max\{P_r \mid 1 \leq r \leq N \& R_r \geq R\}. \tag{38}$$

The *11-point average precision* (Witten, Moffat, & Well, 1999) is defined as the average of IP_R over the eleven R values $0, 0.1, 0.2, \ldots, 0.9, 1.0$. In all the results that we will report here we have applied the 11-point average precision as a summary statistic by which to judge performance. There is one minor problem. The score is used to rank but in some cases the same score is repeated for a number of ranks. In that case the score does not completely determine the rank. Our approach is to divide the total number of objects labeled $+1$ in those ranks by the number of ranks and consider this quotient as the precision at each of the ranks. For example if three objects labeled $+1$ occur over ten ranks with the same score we consider that 0.3 objects labeled $+1$ occur at each of the ten ranks. Finally, training gives explicit values for p_t on the set $h_t[X]$. We use the interpolation of 2.2 to extend p_t (and hence k_t) to new values. In this way the hypothesis $f(x)$ is defined over the entire space.

4. The PAV training advantage

From Theorem 2 we can expect that PAV-AdaBoost enjoys a training advantage over linear AdaBoost. This is not just an advantage in the speed with which PAV-AdaBoost learns but also in an absolute sense. PAV-AdaBoost can learn things (about the training set) that linear AdaBoost cannot learn at all. The purpose of this section is to give simple examples illustrating this fact. Because we are not concerned here with the generalizability of learning, but rather with what can be learned about the training set, we will not need to consider a test space separate from the training space. The examples will involve a training space and we will examine the performance on the training space.

One setting in which we find boosting useful is to combine several different scorings in an attempt to produce a scoring superior to any one of them. Suppose we are given a labeled training set X and a set of scoring functions $\{s_t\}_{t=1}^T$ where each scoring function tends to associate higher scores with the $+1$ label. Then we can apply the PAV-AdaBoost.finite algorithm, 3.3, to combine the scores $\{s_t\}_{t=1}^T$ to produce a final output score f as defined in Eq. (15). Whether this is successful will depend on the scores $\{s_t\}_{t=1}^T$, but we may expect

some success if they come from different sources or express different aspects of the data that are important for distinguishing the two classes. For comparison purposes we also optimize linear AdaBoost by repeated application of the algorithm to the same limited set of scores (weak hypotheses) until convergence. In the case of linear AdaBoost convergence is assured by a slight modification of the proof of Theorem 3 or by Collins, Schapire, and Singer (2002). We find there are many examples where PAV-AdaBoost is successful but linear AdaBoost is not.

4.1. Example

We assume that the training space X consists of four mutually disjoint subsets A, B, C, and D each of which contains 100 objects. Table 1 gives the details. If we assume a scoring function will take a constant value on each of the subsets, then the best possible scoring function will order the sets in increasing order as they appear in the first row of the table. For such a scoring function the precision (11-point average) will be 0.860. When PAV-AdaBoost.finite is applied to this example in one round (processing s_1 and s_2 in turn) it produces an f giving this ideal order and a precision of 0.860. However, linear AdaBoost produces a precision of 0.823. In fact it is clear that linear AdaBoost can never give the ideal ordering because in $\alpha_1 s_1 + \alpha_2 s_2$, we must have $\alpha_1 > \alpha_2$ in order to score points in D above points in C, but then points in A will necessarily score above points in B unless both α's are negative. But if both α's are negative, then B points will score above D points.

4.2. Example

Here we use a training set X similar to that used in the previous example composed of four mutually disjoint subsets A, B, C, and D each of which contains 100 objects. The details are given in Table 2. The ideal order for this space is D at the top followed by B and C in

Table 1. Training set $X = A \cup B \cup C \cup D$. Two scoring functions are defined, but no linear combination of them can order the sets in the order of the probability of $+1$ labels.

Subset	A	B	C	D
$+1$ labels	1	50	51	99
Score s_1	$s_1 = 3$	$s_1 = 1$	$s_1 = 4$	$s_1 = 5$
Score s_2	$s_2 = 1$	$s_2 = 2$	$s_2 = 4$	$s_2 = 3$

Table 2. Training set $X = A \cup B \cup C \cup D$. Two scoring functions are defined, and a linear combination of them can produce the ideal order of the probability of $+1$ labels.

Subset	A	B	C	D
$+1$ labels	0	20	20	80
Score s_1	$s_1 = 0$	$s_1 = 1$	$s_1 = 0$	$s_1 = 1$
Score s_2	$s_2 = 0$	$s_2 = 0$	$s_2 = 1$	$s_2 = 1$

either order and lowest A. This order is produced in one round by PAV-AdaBoost.finite with a precision of 0.698. When linear AdaBoost is applied the α's produced are zero leading to a random performance with a precision of 0.300.

4.3. Example

Here we consider a more conventional problem, namely, boosting naïve Bayes'. We use the binary or multivariate form as our weak learner, though superior performance has been claimed for the multinomial form of naïve Bayes' (McCallum & Nigam, 1998). We find the binary form more convenient to use for simulated data as one need not assign a frequency to a feature in a document, but simply whether the feature occurs. Our goal is not the best absolute performance, but rather to compare different approaches to boosting. For simplicity we will consider a document set that involves only three words. We assume any document has at least one word in it. Then there are seven kinds of documents: $\{x, y, z\}$, $\{x, y\}$, $\{x, z\}$, $\{y, z\}$, $\{x\}$, $\{y\}$, $\{z\}$. We assume that the training space X consists of 700 documents, 100 identical documents of each of these types. To complete the construction of the training space we randomly sample a number from a uniform distribution on the 101 numbers from 0 to 100 inclusive and assign the $+1$ label that many times. This is repeated for each of the seven document types to produce the labeling on all of X. Such a random assignment of labels may be repeated to obtain as many different versions of X as desired. We repeated this process 1000 times and learned on each of these spaces with several different algorithms and the results in summary are given in Table 3. The different methods of learning examined are naïve Bayes', PAV-AdaBoost, and linear AdaBoost with naïve Bayes' as weak learner, and lastly stump AdaBoost. Stump AdaBoost is boosting with decision trees of depth 1 as weak learner (Carreras & Marquez, 2001; Schapire & Singer, 1999). We included stump AdaBoost because it, along with Bayes' and linear AdaBoost, is a linear classifier which produces a weighting of the individual features. On the other hand PAV-AdaBoost is nonlinear. The ideal category in the table is the result if document groups are ordered by the probability of the $+1$ label. The random category is the precision

Table 3. Performance comparisons between different classification methods on 1000 simulated training spaces. The second column gives the average over all training spaces of the 11-point average precision for each of the different methods. The counts listed in a given row and column give the number of training spaces on which the performance of the method listed on that row outperformed/equaled/fell below the method listed at the top of that column.

	11-pap	PAV-Ada	LinAda	StmpAda	Bayes'
Ideal	0.7721	962/38/0	993/7/0	990/10/0	993/7/0
PavAda	0.7321		949/41/10	885/37/78	954/44/2
LinAda	0.7053	10/41/949		220/534/246	67/901/32
StmpAda	0.7047	78/37/885	246/534/220		284/494/222
Bayes'	0.7042	2/44/954	32/901/67	222/494/284	
Random	0.5039	0/0/1000	0/0/1000	0/0/1000	0/0/1000

produced by dividing the total number of +1 labels by 700. Examination of the data shows that all the classifiers perform well above random. While PAV-AdaBoost seldom reaches the ideal limit, in the large majority of training spaces it does outperform the other classifiers. However, one cannot make any hard and fast rules, because each classifier outperforms every other classifier on at least some of the spaces generated.

5. The learning capacity and generalizability of PAV-AdaBoost

The previous examples suggest that PAV-AdaBoost is a more powerful learner than linear AdaBoost and the question naturally arises as to the capacity of PAV-AdaBoost and whether PAV-AdaBoost allows effective generalization. First, it is important to note that the capacity of the individual hypothesis $h(x)$ to shatter a set of points is never increased by the transformation to $k(h(x))$ where k comes from the isotonic regression of +1 labels on $h(x)$. If h cannot distinguish points, then neither can h composed with a monotonic non-decreasing function. However, what is true of the output of a single weak learner is not necessarily true of the PAV-AdaBoost algorithm. In the appendix we show that the hypothesis space generated by PAV-AdaBoost.finite, based on the PAV algorithm defined in 2.1 and the interpolation 2.2, has in many important cases infinite VC dimension. For convenience we will refer to this as the *pure* form of PAV-AdaBoost. In the remainder of this section we will give what we will refer to as the *approximate* form of PAV-AdaBoost. This approximate form produces a hypothesis space of bounded VC dimension and allows a corresponding risk analysis, while yet allowing one to come arbitrarily close to the pure PAV-AdaBoost.

Our approximation is based on a finite set of intervals $B_t = \{I_t^j\}_{j=1}^{n_t}$ for each t, which partition the real line into disjoint subsets. For convenience we assume these intervals are arranged in order negative to positive along the real line by the indexing j. The B_t will be used to approximate the $p_t(s)$(defined earlier by 2.1 and 2.2) corresponding to a weak hypothesis h_t. For each t the set B_t is assumed to be predetermined independent of what sample may be under consideration. Generally in applications there is enough information available to choose B_t intelligently and to obtain reasonable approximations.
Assuming such a predetermined $\{B_t\}_{t=1}^T$, then given $\{h_t\}_{t=1}^T$ define

$$bh_t(x) = j, \quad h_t(x) \in I_t^j, \quad 1 \le j \le n_t, \quad 1 \le t \le T. \tag{39}$$

The result is a new set of hypotheses, $\{bh_t\}_{t=1}^T$, that approximate $\{h_t\}_{t=1}^T$. We simply apply the PAV-AdaBoost algorithms to this approximating set. The theorems of Section 3 apply equally to this approximation. In addition we can derive bounds on the risk or expected testing error for the binary classification problem. First we have a bound on the VC dimension of the hypothesis space.

Theorem 4. *Let $\{B_t\}_{t=1}^T$ denote the set of partitions to be applied to T hypotheses $\{h_t\}_{t=1}^T$ defined on a space X as just described. Let $\mathcal{H}$ denote the set of all hypotheses $H(x) = \operatorname{sign}\left(\sum_{t=1}^T k_{\lambda t}(bh_t(x))\right)$ defined on X where $\{k_{\lambda t}\}_{t=1}^T$ varies over all sets of monotonically nondecreasing functions whose ranges are within $[-\lambda, \lambda]$. Then the VC dimension of $\mathcal{H}$ satisfies*

$$\mathrm{VC} \leq \prod_{t=1}^{T} n_t. \tag{40}$$

Proof: Define sets

$$A_{r_1,r_2,\ldots,r_T} = \bigcap_{t=1}^{T} bh_t^{-1}[r_t], \quad 1 \leq r_t \leq n_t, \quad 1 \leq t \leq T. \tag{41}$$

Note that these sets partition the space X and that any function of the form $H(x) = \mathrm{sign}\big(\sum_{t=1}^{T} k_{\lambda t}(bh_t(x))\big)$ is constant on each such set. Then it is straightforward for the family of all such functions $H(x)$ that we have the stated bound for the VC dimension. □

Vapnik (1998), p. 161, gives the following relationship between expected testing error (R) and empirical testing error (R_{emp}).

$$R \leq R_{\mathrm{emp}} + \frac{E(N)}{2}\left(1 + \sqrt{1 + \frac{R_{\mathrm{emp}}}{E(N)}}\right) \tag{42}$$

which holds with probability at least $1 - \eta$, where $E(N)$ depends on the capacity of the set of functions and involves η and the sample size N.

Corollary 2. *Let $\{B_t\}_{t=1}^{T}$ denote the set of partitions to be applied to T hypotheses. If S is any random sample of points from X of size $N > \prod_{t=1}^{T} n_t$ and $\eta > 0$ and if $H(x) = \mathrm{sign}\big(\sum_{t=1}^{T} k_{\lambda t}(bh_t(x))\big)$ represents T rounds of PAV-AdaBoost learning or the result of AdaBoost.finite applied to S, then with probability at least $1 - \eta$ the risk (R) or expected test error of H satisfies inequality*

$$R \leq R' + \frac{1}{2}\big(E' + \sqrt{E'(E' + R')}\big) \tag{43}$$

where

$$E' = \frac{4}{N}\left[\prod_{t=1}^{T} n_t\big(1 + \log(2N) - \sum_{t=1}^{T} \log(n_t)\big)\big) - \log(\eta/4)\right] \tag{44}$$

and

$$R' = \prod_{t=1}^{T} Z_t \tag{45}$$

in the case of PAV-AdaBoost or equivalently

$$R' = N^{-1} \sum_{x \in S} \exp\left[-y(x) \sum_{t=1}^{T} k_{\lambda t}(h_t(x)) \right] \tag{46}$$

in the case of PAV-AdaBoost.finite.

Proof: First note that relation (42) can be put in the form (43) by a simple algebraic rearrangement where $E(N)$ corresponds to E' and R_{emp} corresponds to R'. It is evident that the right side of (43) is an increasing function of both E' and R' when these are nonnegative quantities. Vapnik (1998) , p. 161, gives the relationship

$$E(N) = \frac{4}{N} \left\{ VC\left[\ln\left(\frac{2N}{VC} \right) + 1 \right] - \ln\left(\frac{\eta}{4} \right) \right\}. \tag{47}$$

The relation (43) then follows from inequality (40) and the bound on N, which insures that $E(N) \leq E'$ and the fact that $R_{\text{emp}} \leq R'$. The latter is true because relations (45) and (46) are simply different expressions for the same bound on the training error (empirical error) for PAV-AdaBoost, but only the second is defined for PAV-AdaBoost.finite. □

In practice we use approximate PAV-AdaBoost with a fine partitioning in an attempt to minimize R'. This does not benefit E' but we generally find the results quite satisfactory. Two factors may help explain this. First, when the PAV algorithm is applied pooling may take place and effectively make the number of elements in a partition B_t smaller than n_t. Second, bounds of the form (43) are based on the VC dimension and ignore any specific information regarding the particular distribution defining a problem. Thus tighter bounds are likely to hold even though we do not know how to establish them in a particular case (Vapnik, 1998). This is consistent with (Burges, 1999; Friedman, Hastie, & Tibshirani, 2000) where it is pointed out that in many cases bounds based on the VC dimension are too loose to have practical value and performance is much better than the bounds would suggest. However, we note that even when N is not extremely large, E' can be quite small when T is small. Thus the relation (43) may yield error bounds of some interest when PAV-AdaBoost.finite is used to combine only two or three hypotheses.

6. Combining scores

We have applied PAV-AdaBoost to the real task of identifying those terms (phrases) in text that would not make useful references as subject identifiers. This work is part of the electronic textbook project at the National Center for Biotechnology Information (NCBI). Textbook material is processed and phrases extracted as index terms. Special processing is done to try to ensure that these phrases are grammatically well formed. Then the phrases must be evaluated to decide whether they are sufficiently subject specific to make useful reference indicators. Examples of phrases considered useful as index terms are *muscles,*

Table 4. Performance of individual scoring methods designed to provide information on a terms usefulness as a reference link to textbook material. The random entry at the bottom simply shows the proportion of terms marked as not useful by the two judges.

Scoring method	11-pap, *A*	11-pap, *B*
s_1 =-strength of context	0.4737	0.2918
s_2 =-database comparison	0.4198	0.2505
s_3 =-number of words	0.2240	0.1373
s_4 =-number of letters	0.2548	0.1500
s_5 = ambiguity	0.2884	0.1907
s_6 = -coherence	0.2932	0.1882
random	0.2031	0.1272

adenosine, beta2 receptors, cerebral cortex, fibroblasts, and *vaccine.* Examples of terms considered too nonspecific are *concurrent*, *brown*, *dilute*, *worsening*, *suggestive*, *cent*, and *roof*. The specific terms are used to index the text in which they occur. If a PubMed® user is looking at a MEDLINE® record and wishes more information about the subject of the record he can click a button that marks up the record with hotlinks consisting of all those specific phrases that occur in the text and also occur as index terms to book sections. If he follows one of these links he will be allowed to choose from, usually several, textbook sections that are indexed by the selected term. By following such links he may gain useful information about the subject of the PubMed record.

In order to study the problem of identifying those terms that are not useful as subject indicators or reference markers, we randomly selected 1000 PubMed records and had two judges with biological expertise examine each marked up record. The judges worked independently and selected any terms that they considered unsuitable for reference and marked them so. This resulted in two sets of judgments which we will refer to as *A* and *B*. The number of terms judged totaled 11,721. Independently of the work of these judges we produced several different scorings of the same set of terms. These scorings are detailed in Table 4. The methods for calculating the strength of context and database comparison scores are described in Kim and Wilbur (2001).

Briefly the more context is associated with a term the more likely that term is useful in representing the subject. The minus sign is because we are trying to identify the terms that are not useful. The database comparison score comes from comparing the frequency of a term in MEDLINE with its occurrence in a database not focused on medicine. If the term is seen much more frequently in a medical database it is more likely to be useful. Again the sign is switched because of our negative purpose. Terms with more words or more letters tend to be more specific and hence more likely to be useful. Ambiguity is a measure of how frequently the term co-occurs with two other terms significantly when these two terms tend to avoid each other. Terms that are ambiguous tend to be poor subject indicators. Likewise coherence is a method of assessing the extent to which terms that co-occur with the given term imply each other. It is computed somewhat differently than ambiguity and does not have the same performance.

Table 5. The 11-point average precisions of different classifiers tested through 10-fold cross validation separately on judgment sets A and B.

Train/test	Mahalanobis	Log-linear	Cmls	Lin-Ada	PAV-Ada
A/A	0.5402	0.5667	0.5634	0.5140	0.6023
B/B	0.3461	0.3632	0.3614	0.3520	0.3640

We applied five different methods to combine these scores to produce a superior scoring of the data. In addition to PAV-AdaBoost.finite and linear AdaBoost we applied Mahalanobis distance (Duda, Hart, & Stork, 2000), a log-linear model (Johnson et al., 1999), and the Cmls wide margin classifier described by Zhang and Oles (2001). For judge A we applied all methods in a 10-fold cross validation to see how well they could learn to predict A's judgments on the held out test material. The same analysis was repeated based on B's judgments. Results for the different methods applied to the two judgement sets, A and B, are given in Table 5. The log-linear, Cmls, and linear AdaBoost methods are linear classifiers. Mahalanobis distance is nonlinear but designed to fit multivariate normal distributions to the $+1$ and -1 labeled classes and use the optimal discriminator based on these fittings. None of these methods work as well as PAV-AdaBoost.finite.

In order to examine the performance of PAV-AdaBoost.finite in another environment, we simulated two three dimensional multivariate normal distributions. Distribution N_1 was derived from the covariance matrix

$$\begin{pmatrix} 2 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

and the points obtained were labeled as $+1$. Distribution N_{-1} was derived from the covariance matrix

$$\begin{pmatrix} 6 & 5 & 5 \\ 5 & 6 & 5 \\ 5 & 5 & 6 \end{pmatrix}$$

and points were labeled as -1. Five hundred points were sampled from each distribution to produce a training set of 1000 points. Another 1000 points were sampled in the same manner to produce a test set. The location of the means for distribution N_1 and N_{-1} were varied to be either closer to each other or farther apart to produce learning challenges of varying difficulty. Training and testing sets of three levels of difficulty were produced and the same five classifiers applied to the previous data set were applied here. The results are given in Table 6. Sets 1, 2, and 3 are of increasing level of difficulty. Since this problem is ideal for the Mahalanobis distance method, we are not surprised to see that it outperforms the other methods in all cases. However, among the other four methods PAV-AdaBoost.finite gives

Table 6. Eleven point average precisions for classifiers trained to discriminate between two three-dimensional normal distributions. Training took place on one sample and testing on an independently generated sample from the same distributions. The weak learners for boosting are the three co-ordinate projections.

	N_1 mean	N_2 mean	Mahalanobis	Log-linear	Cmls	Lin-Ada	PAV-Ada
Set 1	$(-1, -1, -1)$	(2, 2, 2)	0.8556	0.7973	0.8235	0.8341	0.8357
Set 2	(0,0,0)	(1, 1, 1)	0.7250	0.5980	0.5961	0.5950	0.6107
Set 3	(0,0,0)	(1/2, 1/2, 1/2)	0.7014	0.5653	0.5606	0.5568	0.5624

the best performance on two cases and is a close second to the Log-Linear method on the third case.

7. Document classification

We are concerned with the rating of new documents that appear in a large database (MEDLINE) and are candidates for inclusion in a small specialty database (REBASE®). The requirement is to rank the new documents as nearly in order of decreasing potential to be added to the smaller database as possible, so as to improve the coverage of the smaller database without increasing the effort of those who manage this specialty database. To perform this ranking task we have considered several machine learning approaches and have developed a methodology of testing for the task. The version of REBASE (a restriction enzyme database) we study here consists of 3,048 documents comprising titles and abstracts mostly taken from the research literature. These documents are all contained in MEDLINE and we have ignored that small part of the database not contained in MEDLINE. We have applied naïve Bayes' (binary form) to learn the difference between REBASE and the remainder of MEDLINE and extracted the top scoring 100,000 documents from MEDLINE that lie outside of REBASE. We refer to this set as NREBASE. These are the 100,000 documents which are perhaps most likely to be confused with REBASE documents. We have set ourselves the task of learning to distinguish between REBASE and NREBASE. In order to allow testing we randomly sampled a third of REBASE and called it RTEST and a third of NREBASE and called it NRTEST. We denote the remainder of REBASE as RTRAIN and the remainder of NREBASE as NRTRAIN. Thus we have training and testing sets defined by

$$\begin{aligned} \text{TRAIN} &= \text{RTRAIN} \cup \text{NRTRAIN} \\ \text{TEST} &= \text{RTEST} \cup \text{NRTEST} \end{aligned} \tag{48}$$

We have applied some standard classifiers to this data and obtained the results contained in Table 7. Here naïve Bayes' is the binary form and we discard any terms with weights in absolute value less than 1.0. Such a cutting procedure is a form of feature selection that often gives improved results for naïve Bayes'. We also use the binary vector representation for Cmls (Zhang & Oles, 2001) and use the value 0.001 for the regularization parameter λ

Table 7. Results of some standard classifiers on the REBASE learning task.

Classifier	Naïve Bayes'	KNN	Cmls
11-pap	0.775	0.783	0.819

as suggested in Zhang and Oles, (2001). KNN is based on a vector similarity computation using *IDF* global term weights and a *tf* local weighting formula. If *tf* denotes the frequency of term t within document d and *dlen* denotes the length of d (sum of all tf' for all t'in d) then we define the local weight

$$lw_{tf} = \frac{1}{1 + \lambda^{tf-1} \exp(\alpha \cdot dlen)} \tag{49}$$

where $\alpha = 0.0044$ and $\lambda = 0.7$. This formula is derived from the Poisson model of term frequencies within documents (Kim, Aronson, & Wilbur, 2001; Robertson & Walker, 1994) and has been found to give good performance on MEDLINE documents. The similarity of two documents is the dot product of their vectors and no length correction is used as that is already included in definition (49). KNN computes the similarity of a document in the test set to all the documents in the training set and adds up the similarity scores of all those documents in the training set that are class $+1$ (in RTRAIN in this case) in the top K ranks. We use a K of 60 here.

One obvious weak learner amenable to boosting is naïve Bayes'. We applied linear AdaBoost and obtained a result on the test set of 0.783 on the third round of boosting. With naïve Bayes' as the weak learner PAV-AdaBoost does not give any improvement. We will discuss the reason for this subsequently. Our interest here is to show examples of weak learners where PAV-AdaBoost is successful.

One example where PAV-AdaBoost is successful is with what we will call the Cmass weak learner.

7.1. Cmass algorithm

Assume a probability distribution $D(i)$ on the training space $X = \{(\vec{x}_i, y_i)\}_{i=1}^m$. Let P denote that subset of X with $+1$ labels and N the subset with -1 labels. To produce h perform the steps:

(i) Use $D(i)$ to weight the points of P and compute the centroid $\vec{c}_p$.
(ii) Similarly weight the points of N and compute the centroid $\vec{c}_n$.
(iii) Define $\vec{w} = \vec{c}_p - \vec{c}_n$ and $h(\vec{x}) = \vec{w} \cdot \vec{x}$ for any $\vec{x}$.

With the Cmass weak learner PAV-AdaBoost gives the results depicted in Figure 1. PAV-AdaBoost reaches a peak score just above 0.8 after about 60 iterations. Since Cmass is very efficient to compute it takes just under eleven minutes to produce one hundred rounds

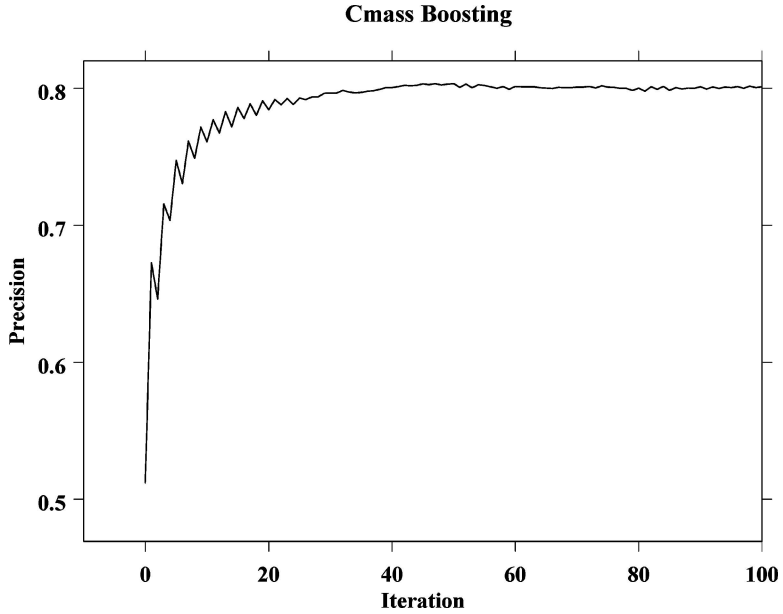

Figure 1. PAV-AdaBoost with Cmass as weak learner. Precision is 11-point average precision.

of boosting on a single 500 MH Pentium Xeon processor. There is some similarity here to wide margin classifiers such as SVM or Cmls in that those data points that are easily classified are progressively ignored (are not support vectors) as the processing continues to focus on those vectors which are eluding the classifier. Linear AdaBoost fails to boost with Cmass. It does this by first producing an alpha of approximately 0.86 and then proceeding to produce negative alphas which converge rapidly to zero. The scores produced in these steps actually decrease slightly from the initial Cmass score of 0.512.

Another weak learner where PAV-AdaBoost is able to boost effectively is what we term Nscore (neighbor score).

7.2. *Nscore algorithm*

Assume a probability distribution $D(i)$ on the training space $X = \{(\vec{x}_i, y_i)\}_{i=1}^m$. Let P denote that subset of X with $+1$ labels and N the subset with -1 labels. Let $t-1$ denote the number of times we have already applied the algorithm. Then if t is even choose the member $\vec{x}_i$ of P with greatest $D(i)$ (or one of the greatest in a tie). If t is odd, make the same choice from N instead. If t is even define $h_t(\vec{x}) = \text{sim}(\vec{x}_i, \vec{x})$, while if t is odd define $h_t(\vec{x}) = -\text{sim}(\vec{x}_i, \vec{x})$, for any $\vec{x}$. Here $\text{sim}(\vec{x}_i, \vec{x})$ is the similarity computation used in KNN above and defined in (49) and by *IDF* weighting.

PAV-AdaBoost applied with Nscore as weak learner reaches a score of 0.822 at around 4,000 iterations and remains at 0.82 at 10,000 iterations. The 4,000 iterations take 30 minutes

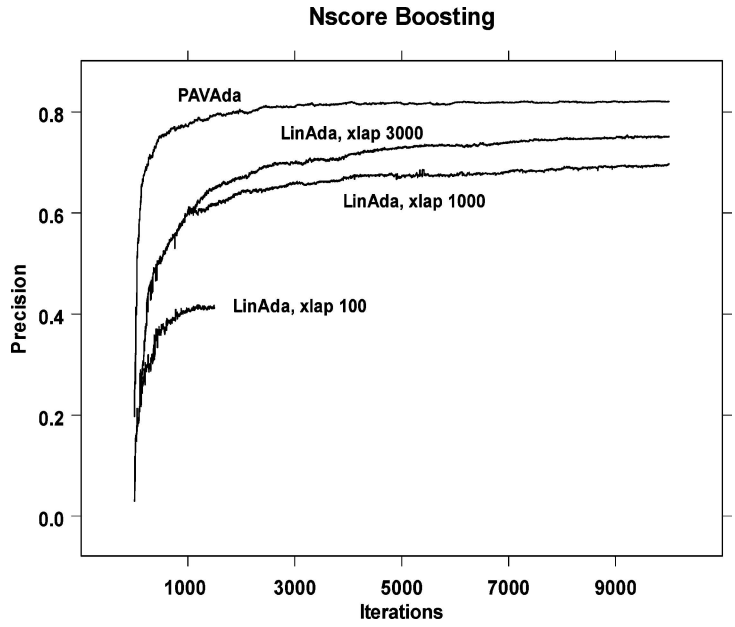

Figure 2. Boosting Nscore with PAV-AdaBoost compared with several different attempts to boost Nscore with linear AdaBoost.

on a single 500 MH Pentium Xeon processor. The result is shown in Figure 2. Attempts to apply linear AdaBoost with Nscore as defined in 7.2 fail. The program locks onto a small subset of the space and repeats these elements as the alphas move to zero and boosting becomes ineffective. In order to perform linear AdaBoost effectively we added a parameter that would only allow Nscore to repeat a point after *xlap* iterations. We tried values for *xlap* of 100, 1,000, and 3,000. The results are shown in Figure 2. While PAV-AdaBoost clearly peaks in the first 10,000 iterations, linear AdaBoost does not. We continued the linear AdaBoost ($xlap = 3{,}000$) calculation for a total of 50,000 iterations and reached a final score of 0.795. During the last 6,000 iterations it did not increase its peak level so that it appears nothing more could be gained by continuing an approximately day long calculation.

8. Discussion

We are not the first to attempt a reweighting of weak hypotheses in boosting depending on an assessment of the quality of training for different regions of the data. Several have done this based on properties of the data which are outside of or in addition to the information to be gained from the values of the individual weak hypotheses themselves (Maclin, 1998; Meir, El-Yaniv, & Ben-David, 2000; Moerland & Mayoraz, 1999). The bounds on expected

testing error in (Meir, El-Yaniv, and Ben-David, 2000) are based on a partition of the space and show a dependency on the dimension of that space very similar to our results in Theorem 4 and Corollary 2. Our general approach, however, is more closely related to the method of Aslam (2000) in that our reweighting depends only on the relative success of each weak hypothesis in training. Aslam evaluates success on the points with positive and negative label independently and thereby decreases the bound on training error. In a sense we carry this approach to its logical limit and decrease the bound on the training error as much as possible within the paradigm of confidence rated predictions. Our only restriction is that we do not allow the reweighting to reverse the order of the original predictions made by a weak learner. Because our approach can be quite drastic in its effects there is a tendency to over train. Corollary 2 suggests that in the limit of large samples results should be very good. However in practical situations one is forced to rely on experience with the method. Of course this is generally the state of the art for all machine learning methods.

An important issue is when PAV-AdaBoost can be expected to give better performance than linear AdaBoost. The short answer to this question is that all the examples in the previous two sections where PAV-AdaBoost is quite successful involve weak hypotheses that are intrinsic to the data and not produced by some optimization process applied to a training set. As an example of this distinction let the entire space consist of a certain set X of documents and let w be a word that occurs in some of these documents but not others. Suppose that some of these documents that are about a particular topic have the $+1$ label and the remainder the -1 label. Then the function

$$h_w(x) = \begin{cases} 1, & w \in x \\ 0, & w \notin x \end{cases} \tag{50}$$

is what we are referring to as an intrinsic function. Its value does not depend on a sample from X. On the other hand if we take a random sample $Y \subseteq X$ which is reasonably representative of the space, then we can define the Bayesian weight b_w associated with w. If we define the function

$$h'_w(x) = \begin{cases} b_w, & w \in x \\ 0, & w \notin x \end{cases} \tag{51}$$

Then it is evident that h'_w depends on the sample as well as the word w. It is not intrinsic to the space X of documents. We believe this distinction is important as the non-intrinsic function h'_w is already the product of an optimization process. Since PAV is itself an optimization process it may be that applying one optimization process to the output of another is simply too much optimization to avoid over training in most cases.

We believe naïve Bayes' provides an instructive example in discussing this issue. Bayesian scores produced on a training set do not correspond to an intrinsic function because the same data point may receive a different score if the sample is different. Thus it is evident that there is already an optimization process involved in Bayesian weighting and scoring. In applying PAV-AdaBoost with naïve Bayes' as weak learner we find poor performance. Because PAV-AdaBoost is a powerful learner, poor generalizability is best understood as

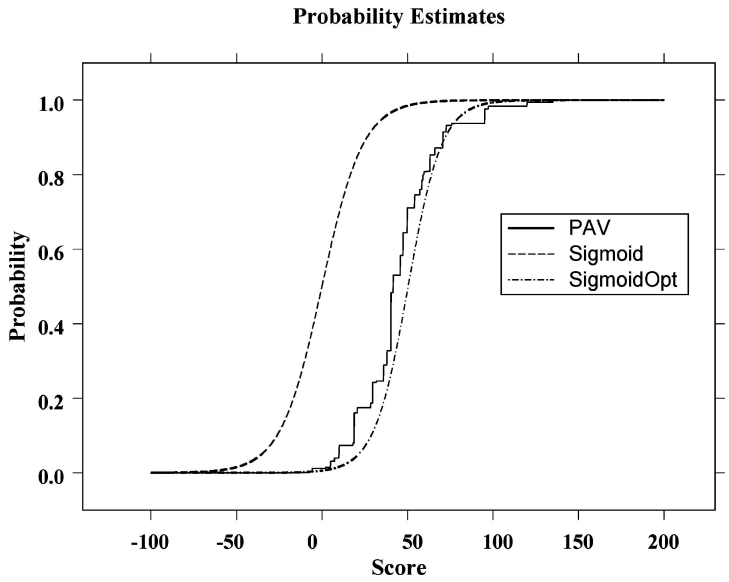

Figure 3. Probability of label class +1 as a function of score as estimated by PAV and by sigmoid curves. The curve marked Sigmoid is the estimate implicitly used by linear AdaBoost and is clearly not the optimal sigmoid minimizing Z_t. SigmoidOpt is the result of optimization over both w and b in definition (52).

a consequence of overtraining. In the case of PAV-AdaBoost applied to naïve Bayes' such overtraining appears as PAV approximates a generally smooth probability function with a step function that reflects the idiosyncrasies of the training data in the individual steps. We might attempt to overcome this problem by replacing the step function produced by PAV by a smooth curve. The PAV function derived from naïve Bayes' training scores (learned on TRAIN) and a smooth approximation are shown in Figure 3. The smooth curve has the formula

$$\sigma(s) = \left(1 + e^{-(ws-b)}\right)^{-1} \tag{52}$$

and is the sigmoid shape commonly used in neural network threshold units (Mitchell, 1997). The parameters w and b have been optimized to minimize Z. If we use this function as we do $p(s)$ in definition (14) we obtain

$$k(s) = \frac{1}{2}\ln\left(\frac{\sigma(s)}{1-\sigma(s)}\right) = \frac{1}{2}(ws - b). \tag{53}$$

Thus if we set $b = 0$ and optimize on w alone to minimize Z, we find that the function (52) gives the probability estimate used implicitly by linear AdaBoost with $\alpha = w/2$. The resultant sigmoid curve is also shown in Figure 3. From this picture it is clear that neither PAV-AdaBoost nor linear AdaBoost are ideal for boosting naïve Bayes'. PAV-AdaBoost

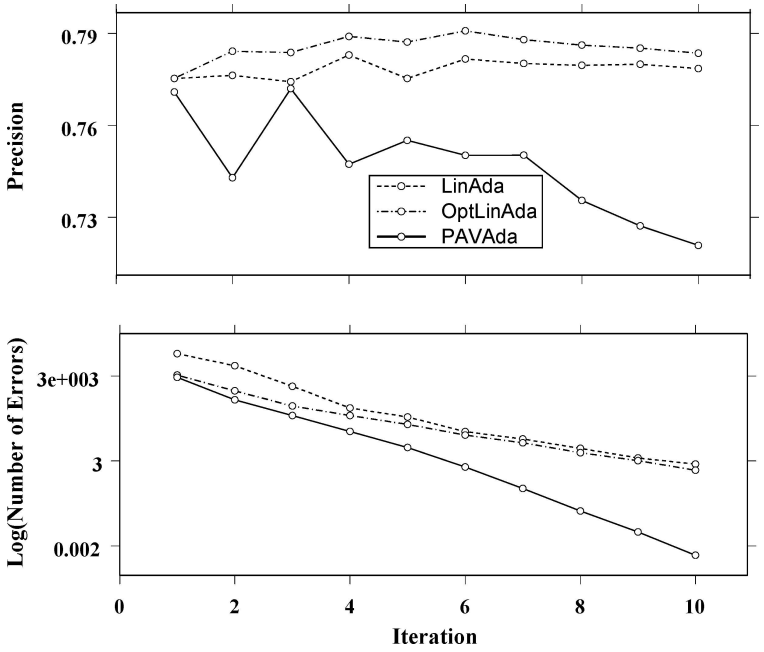

Figure 4. Boosting naïve Bayes (binary form) with three different algorithms. PAV-AdaBoost gives the lowest error on the training space (lower panel), but optimal linear AdaBoost gives the best generalizability (upper panel).

because it over trains and linear AdaBoost because it is too far from optimal. A better choice would be what we may call optimal linear AdaBoost which is the result of optimizing formula (52) over both w and b. The results of these three different strategies applied to boost naïve Bayes' are shown in Figure 4. While optimal linear AdaBoost outperforms the other methods on this problem, it is evident that none of the methods perform very well. We attribute this to the fact that once individual term weights are added into the Bayes' score it is not possible for Boosting to ameliorate the effects of the dependency between terms. It has been known for some time (Langley & Sage, 1994) that dealing with inter-term dependencies can improve the performance of naïve Bayes'. We hypothesize that to be most effective this must take place at the level of individual terms.

Let us return to the main issue here, namely, when is linear AdaBoost preferable to PAV-AdaBoost. In examining Figure 3 it is not obvious that linear AdaBoost would perform better than PAV-AdaBoost. Each method fails to be optimal. In order to obtain more insight we repeatedly resampled the training and testing sets and performed boosting by the two methods. We found that generally linear AdaBoost outperformed PAV-AdaBoost. It was common that neither method worked very well, but generally linear AdaBoost had the edge. This leads us to the general conclusion that when a weak learner produces a logodds score linear AdaBoost may be a good choice for boosting. Of course in this situation optimal linear AdaBoost may be even better. When the score is not a logodds score a sigmoid curve may not give a good approximation and in that situation PAV-AdaBoost may give the better result.

Finally, there is the question of why PAV-AdaBoost applied to Nscore produces such a good result. The score of 0.822 is superior to the results in Table 7. We have only been able to produce a higher score on this training-testing pair by applying AdaBoost to optimally boost over 20,000 depth four decision trees. By this means we have obtained a score just larger than 0.84, but the computation required several days. While this is of theoretical interest it is not very efficient for an approximately 2% improvement. We believe the good result with PAV-AdaBoost applied to Nscore is a consequence of the fact that we are dealing with a large set of relatively simple weak learners that have not undergone prior optimization. We see the result of PAV-AdaBoost applied to Nscore as a new and improved version of the K-nearest neighbors algorithm.

Appendix

Here we examine the VC dimension of the set of hypotheses generated by the PAV-AdaBoost.finite algorithm based on the pure form of PAV defined by 2.1. This presupposes a fixed set of weak hypotheses $\{h_t\}_{t=1}^T$ and the set of functions under discussion is all the functions $H(x) = \text{sign}(\sum_{t=1}^T k_{\lambda t}(h_t(x)))$ obtained by applying PAV-AdaBoost.finite to possible finite training samples with any possible labeling of the sample points and then extending the results globally using the interpolation of 2.2. Let us denote this set of functions by $\mathcal{H}$. Different weightings for points are allowed by PAV-AdaBoost.finite, but to simplify the presentation we will assume that all points have weight 1.

To begin it is important to note that the algorithm actually acts on samples of the fixed set of points $\{(h_1(x), h_2(x), \ldots, h_T(x))\}_{x \in X}$ in Euclidean T-space. Thus for this purpose we may disregard the set of weak hypotheses $\{h_t\}_{t=1}^T$ and focus our attention on finite subsets of Euclidean T-space where the role of the weak hypotheses is assumed by the co-ordinate projection functions. We may further observe that whether a set of points can be shattered by $\mathcal{H}$ does not depend on the actual co-ordinate values, but only on the orderings of the points induced from their different co-ordinate values. Finally we observe that if a set of points contains points A and B and A is less than or equal to B in all co-ordinate values, then $\mathcal{H}$ cannot shatter that set of points. The regressions involved must always score B equal to or greater than A if B dominates A. It will prove sufficient for our purposes to work in two dimensions. In two dimensions the only way to avoid having one point dominate another is to have the points strictly ordered in one dimension and strictly ordered in the reverse direction in the other dimension. Thus we may consider a set of N points to be represented by the N pairs $\{(r_i, b - r_{N-i+1})\}_{i=1}^N$ where $\{r_i\}_{i=1}^N$ is any strictly increasing set of real numbers and b is an arbitrary real number that may be determined by convenience. We will refer to the ordering induced by the first co-ordinate as the forward ordering, and the ordering induced by the second co-ordinate as the backward ordering. For a given labeling of such a set of points (which must contain both positive and negative labels) the PAV-AdaBoost.finite algorithm may be applied and will always yield a solution. This solution will depend only on the number of points, the ordering, and the labels, but not on the actual numbers r_i used to represent the points.

In order to shatter sets of points we must examine the nature of solutions coming from PAV-AdaBoost.finite. Clearly for any labeling of the points we may consider the points

as contiguous groups labeled negative or labeled positive and alternating as we move up the forward order. Let the numbers of points in these groups be given by $\{m_i\}_{i=1}^{K}$. We will assume that the lowest group (m_1) in the forward order is labeled negative and the highest group (m_K) is labeled positive. This will allow us to maintain sufficient generality for our purposes. Then K must be an even integer and it will be convenient to represent it by $2k-2$. Our approach will be to assume a certain form for the solution of the PAV-AdaBoost.finite scoring functions and use the invariance of these functions under the algorithm to derive their actual values. For the regression induced function $k_1$obtained from the forward ordering we assume without loss of generality the form

$$k_1(m_p) = \begin{cases} \alpha_1, & p = 1 \\ \alpha_i, & 2i-2 \le p \le 2i-1, \quad 2 \le i \le k-1 \\ \alpha_k, & p = 2k-2 \end{cases} \tag{54}$$

Here $k_1(m_p)$ denotes the value of k_1 on all the points in the group m_p. Likewise the $k_2$obtained from the backward ordering is assumed to have the form

$$k_2(m_p) = \beta_i, \quad 2i-1 \le p \le 2i, \quad 1 \le i \le k-1. \tag{55}$$

By definition (54), α_k corresponds to the single group m_{2k-2} with positive label. As a result of the forward isotonic regression it follows that $\alpha_1 = -\lambda$ and $\alpha_k = \lambda$. Now appealing to the invariance of k_1 and k_2 under the algorithm (apply one round of PAV-AdaBoost.finite and k_1 and k_2 are unchanged) we may derive the relations

$$\begin{aligned} \alpha_j &= -\frac{1}{2}(\beta_{j-1} + \beta_j) + \frac{1}{2}\ln\left(\frac{m_{2j-2}}{m_{2j-1}}\right) \\ \beta_j &= -\frac{1}{2}(\alpha_j + \alpha_{j+1}) + \frac{1}{2}\ln\left(\frac{m_{2j}}{m_{2j-1}}\right) \end{aligned} \tag{56}$$

These relations are valid for α_j, $2 \le j \le k-1$ and for all β_j, $1 \le j \le k-1$. By simply adding to the appropriate expressions in (56) we obtain expressions for the scores over the various m_i.

$$\begin{aligned} \text{score}(m_{2j-1}) = \alpha_j + \beta_j &= \frac{1}{2}(\alpha_j - \alpha_{j+1}) + \frac{1}{2}\ln\left(\frac{m_{2j}}{m_{2j-1}}\right) \\ &= \frac{1}{2}(\beta_j - \beta_{j-1}) + \frac{1}{2}\ln\left(\frac{m_{2j-2}}{m_{2j-1}}\right) \\ \text{score}(m_{2j}) = \alpha_{j+1} + \beta_j &= \frac{1}{2}(\alpha_{j+1} - \alpha_j) + \frac{1}{2}\ln\left(\frac{m_{2j}}{m_{2j-1}}\right) \\ &= \frac{1}{2}(\beta_j - \beta_{j+1}) + \frac{1}{2}\ln\left(\frac{m_{2j}}{m_{2j+1}}\right) \end{aligned} \tag{57}$$

The important observation here is that the odd and even groups will be separated provided the $\alpha_{j+1} - \alpha_j$ are all positive and not overwhelmed by the log terms. If this can be arranged properly it will also guarantee that the regressions are valid as well. By repeated substitutions using Eqs. (57) it can be shown that

$$\alpha_{j+1} - \alpha_j = \frac{1}{k-1}(\alpha_k - \alpha_1) + \frac{1}{k-1}\sum_{p=1}^{k-1}\left[\ln\left(\frac{m_{2j}}{m_{2p}}\right) + \ln\left(\frac{m_{2j-1}}{m_{2p-1}}\right)\right] \tag{58}$$

One crucial result that comes from this relationship is that if all the m_i are equal then all the log terms disappear in Eq. (58) and because $\alpha_1 = -\lambda$ and $\alpha_k = \lambda$ we may conclude from Eqs. (57) that the score of all positive labeled elements is $\lambda/(k-1)$ while the score of all negative labeled elements is $-\lambda/(k-1)$. If the m_i are all equal to two, let us refer to the resulting function $k_1 + k_2$ as a *binary* solution. Such solutions may be used to shatter two dimensional sets of any number of points.

Let $\{s_i\}_{i=1}^N$ be any strictly increasing set of real numbers and $\{(s_i, b - s_{N-i+1})\}_{i=1}^N$ the corresponding set of points in two dimensions. Let a labeling be given that divides the set into contiguous same label groups with the sizes $\{n_j\}_{j=1}^J$ as one moves up the forward ordering. For definiteness let the group corresponding to n_1 have the positive label and the group corresponding to n_J have the negative label. Our strategy is to sandwich each group corresponding to an n_j between two elements that form a binary group in a new system of points for which the binary solution defined above will apply. For any $\varepsilon > 0$ define the strictly increasing set of numbers $\{r_i\}_{i=1}^{2J+4}$ by

$$\begin{aligned}
&r_i = s_1 + (i-4)\varepsilon, \quad i = 1, 2, 3\\
&\left.\begin{aligned} r_{2j+2} &= \frac{2s_k + s_{k+1}}{3}\\ r_{2j+3} &= \frac{s_k + 2s_{k+1}}{3}\end{aligned}\right\}, \quad k = \sum_{q=1}^{j} n_q,\ 1 \le j < J\\
&r_i = s_N + (i - 2J - 1)\varepsilon, \quad i = 2J+2, 2J+3, 2J+4
\end{aligned} \tag{59}$$

Label the two dimensional set of points coming from $\{r_i\}_{i=1}^{2J+4}$ in groups of two starting with negative from the left end and ending in positive at the far right end. Then this set of points has a binary solution and its extension by the interpolation 2.2 separates the positive labeled and negative labeled points of $\{(s_i, b - s_{N-i+1})\}_{i=1}^N$ corresponding to the groupings $\{n_j\}_{j=1}^J$. The construction (59) will vary slightly depending on the labeling of the end groups of $\{(s_i, b - s_{N-i+1})\}_{i=1}^N$, but the same result holds regardless. We have proved the following.

Theorem 5. *For any choice of λ and in any Euclidean space of dimension two or greater, if all points have weight one and the weak hypotheses are the co-ordinate projections, then the VC dimension of the set $\mathcal{H}$ of all hypotheses that may be generated by PAV-AdaBoost.finite over arbitrary samples with arbitrary labelings is infinite.*

The construction of the sets of points $\{(s_i, b - s_{N-i+1})\}_{i=1}^N$ and the set of points corresponding to $\{r_i\}_{i=1}^{2J+4}$ as defined in (59) can all be carried out within any open subset of E^2 for any N by simply choosing ε small enough and adjusting b appropriately.

Corollary 3. *Let $\{h_t\}_{t=1}^T$ denote a fixed set of weak learners defined on a space X and define $U : X \to E^T$ by $U(x) = (h_1(x), h_2(x), \ldots, h_T(x))$. Assume that all points have weight one. If the image of X under U contains an open subset of a 2-dimensional hyperplane parallel to two of the co-ordinate axes in E^T, then for any λ the set of hypotheses coming from PAV-AdaBoost.finite applied to $\{h_t\}_{t=1}^T$ over all possible samples of X and arbitrary labelings has infinite VC dimension.*

Acknowledgments

The authors would like to thank the anonymous referees for very helpful comments in revising this work.

References

Apte, C., Damerau, F., & Weiss, S. (1998). Text mining with decision rules and decision trees. *Conference Proceedings The Conference on Automated Learning and Discovery,* CMU.

Aslam, J. (2000). Improving algorithms for boosting. *Conference Proceedings 13th COLT*. Palo Alto, California.

Ayer, M., Brunk, H. D., Ewing, G. M., Reid, W. T., & Silverman, E. (1954). An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics, 26*, 641–647.

Bennett, K. P., Demiriz, A., & Shawe-Taylor, J. (2000). A column generation algorithm for boosting. *Conference Proceedings 17th ICML.*

Buja, A., Hastie, T., & Tibshirani, R. (1989). Linear smoothers and additive models. *The Annals of Statistics, 17*:2 453–555.

Burges, C. J. C. (1999). *A tutorial on support vector machines for pattern recognition* (Available electronically from the author): Bell Laboratories, Lucent Technologies.

Carreras, X., & Marquez, L. (2001). September 5–7, 2001. Boosting trees for anti-spam email filtering. *Conference Proceedings RANLP2001,* Tzigov Chark, Bulgaria.

Collins, M., Schapire, R. E., & Singer, Y. (2002). Logistic regression, AdaBoost and Bregman distances. *Machine Learning, 48*:1, 253–285.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification* (2 edn.). New York: John Wiley & Sons, Inc.

Duffy, N., & Helmbold, D. (1999). Potential boosters? *Conference Proceedings Advances in Neural Information Processing Systems* 11.

Duffy, N., & Helmbold, D. (2000). Leveraging for regression. *Conference Proceedings 13th Annual Conference on Computational Learning Theory.* San Francisco.

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

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *The Annals of Statistics, 38*:2, 337–374.

Hardle, W. (1991). *Smoothing Techniques: With Implementation in* S. New York: Springer-Verlag.

Johnson, M., Geman, S., Canon, S., Chi, Z., & Riezler, S. (1999). Estimators for stochastic "unification-based" grammars. *Conference Proceedings Proceedings ACL'99.* Univ. Maryland.

Kim, W., Aronson, A. R., & Wilbur, W. J. (2001). Automatic MeSH term assignment and quality assessment. *Conference Proceedings Proc. AMIA Symp.* Washington, D.C.

Kim, W. G., & Wilbur, W. J. (2001). Corpus-based statistical screening for content-bearing terms. *Journal of the American Society for Information Science, 52*:3, 247–259.

Langley, P., & Sage, S. (1994). Induction of selective Bayesian classifiers. *Conference Proceedings Tenth Conference on Uncertainty in Artificial Intelligence,* Seattle, WA.

Maclin, R. (1998). Boosting classifiers locally. *Conference Proceedings Proceedings of AAAI.*

Mason, L., Bartlett, P. L., & Baxter, J. (2000). Improved generalizations through explicit optimizations of margins. *Machine Learning, 38*, 243–255.

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive bayes text classification. *Conference Proceedings AAAI-98 Workshop on Learning for Text Categorization*.

Meir, R., El-Yaniv, R., & Ben-David, S. *2000*. Localized boosting. *Conference Proceedings 13th COLT*. Palo Alto, California.

Mitchell, T. M. (1997). *Machine learning*. Boston: WCB/McGraw-Hill.

Moerland, P., & Mayoraz, E. (1999). *DynamBoost: combining boosted hypotheses in a dynamic way* (Technical Report RR 99-09): IDIAP Switzerland.

Nock, R., & Sebban, M. (2001). A Bayesian boosting theorem. *Pattern Recognition Letters, 22*, 413–419.

Pardalos, P. M., & Xue, G. (1999). Algorithms for a class of isotonic regression problems. *Algorithmica, 23*, 211–222.

Ratsch, G., Mika, S., & Warmuth, M. K. (2001).*On the Convergence of Leveraging (NeuroCOLT2 Technical Report 98)*. London: Royal Holloway College.

Ratsch, G., Onoda, T., & Muller, K.-R. (2001). Soft margins for AdaBoost. *Machine Learning, 42*, 287–320.

Robertson, S. E., & Walker, S. (1994). Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. Conference Proceedings *17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

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

Vapnik, V. (1998). *Statistical Learning Theory*. New York: John Wiley & Sons, Inc.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing Gigabytes (2 edn.)*. San Francisco: Morgan-Kaufmann Publishers, Inc.

Zhang, T., & Oles, F. J. (2001). Text categorization based on regularized linear classification methods. *Information Retrieval*, *4*:1, 5–31.

Received June 20, 2003
Revised March 8, 2005
Accepted March 10, 2005