RUTCOR
RESEARCH
REPORT

Boosting Optimal Logical Patterns Using Noisy Data

Noam Goldberg[a] Chung-chieh Shan[b]

RRR 5-2007, January, 2007

RUTCOR
Rutgers Center for
Operations Research
Rutgers University
640 Bartholomew Road
Piscataway, New Jersey
08854-8003
Telephone: 732-445-3804
Telefax: 732-445-5472
Email: rrr@rutcor.rutgers.edu
http://rutcor.rutgers.edu/∼rrr

[a]RUTCOR, Rutgers, the State University of New Jersey, 640 Bartholomew Road, Piscataway, NJ 08854-8003
[b]Department of Computer Science, Rutgers, the State University of New Jersey, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019

Rutcor Research Report
RRR 5-2007, January, 2007

Boosting Optimal Logical Patterns Using Noisy Data

Noam Goldberg Chung-chieh Shan

Abstract. We consider the supervised learning of a binary classifier from noisy observations. We use smooth boosting to linearly combine abstaining hypotheses, each of which maps a subcube of the attribute space to one of the two classes. We introduce a new branch-and-bound weak learner to maximize the agreement rate of each hypothesis. Dobkin et al. give an algorithm for maximizing agreement with real-valued attributes [9]. Our algorithm improves on the time complexity of Dobkin et al.'s as long as the data can be binarized so that the number of binary attributes is o(log of the number of observations $\times$ number of real-valued attributes). Furthermore, we have fine-tuned our branch-and-bound algorithm with a queuing discipline and optimality gap to make it fast in practice. Finally, since logical patterns in Hammer et al.'s Logical Analysis of Data (LAD) framework [8, 6] are equivalent to abstaining monomial hypotheses, any boosting algorithm can be combined with our proposed weak learner to construct LAD models. On various data sets, our method outperforms state-of-the-art methods that use suboptimal or heuristic weak learners, such as SLIPPER. It is competitive with other optimizing classifiers that combine monomials, such as LAD. Compared to LAD, our method eliminates many free parameters that restrict the hypothesis space and require extensive fine-tuning by cross-validation.

Acknowledgements: This work benefited from discussions with Peter L. Hammer, who passed away prematurely. We also thank Philip M. Long, Robert E. Schapire, Rocco A. Servedio.

1 Introduction

We apply boosting to logical patterns in this paper to learn a binary classifier from noisy data with n Boolean attributes. The training input is a set of observations $\Omega \subseteq \{0,1\}^n \times \{+1,-1\}$, in which each observation (x, y) consists of an n-dimensional Boolean vector of attributes x and a binary classification y. A *logical pattern* p is a conjunction of attributes or their complements. For example, if "male" (x_1) and "married" (x_3) are among the n attributes, then "male and not married" ($x_1 \wedge \bar{x}_3$) is a pattern, but not "male or not married" ($x_1 \vee \bar{x}_3$). A pattern p is equivalent to a subcube Cover(p) of the n-dimensional hypercube $\{0,1\}^n$; we say that p *covers* the Boolean vectors in Cover(p). For example, the pattern $x_1 \wedge \bar{x}_3$ covers the Boolean vector 110 but not 111 or 011. To be learned, a pattern p must cover at least one observed attribute vector; in other words, Cover(p) $\times \{+1,-1\}$ must intersect Ω.

Logical patterns have been considered as rules of machine learning algorithms such as RIPPER and SLIPPER [7]. RIPPER combines logical patterns by set covering and rule ordering. SLIPPER learns rules and combines them into a single hypothesis by boosting, which outputs a linear combination of rules by iterative coordinate descent.

Logical patterns are also a building block of the Logical Analysis of Data (LAD) framework. LAD is a framework for the binary classification of data with Boolean attributes [8, 6].[1] An LAD model specifies a set of patterns $\mathcal{P}$, partitioned into a set of positive patterns $\mathcal{P}^+$ and a set of negative patterns $\mathcal{P}^-$. A positive pattern should cover attribute vectors whose correct classification tends to be positive; analogously for negative patterns. Hence we associate with each pattern p in the model a *weak hypothesis* $h_p : \{0,1\}^n \to \{+1,-1,0\}$, defined by

$$h_p(x) = \begin{cases} +1 & \text{if } p \in \mathcal{P}^+ \text{ and } x \in \text{Cover}(p), \\ -1 & \text{if } p \in \mathcal{P}^- \text{ and } x \in \text{Cover}(p), \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We call p *homogeneous* if it covers positive observations only or negative observations only. Whereas the bulk of the LAD literature [15] requires patterns to be homogeneous, in practice this requirement is often relaxed for noisy data.

To classify new data, an LAD model also defines a discriminant function $\Delta : \{0,1\}^n \to [-1,1]$ by

$$\Delta(x) = \sum_{p \in \mathcal{P}} \alpha_p h_p(x), \tag{2}$$

where α_p are weights normalized to sum to 1. The model then classifies each Boolean vector x as positive or negative according to the sign of $\Delta(x)$.

To build an LAD model, we can weigh the patterns in a variety of ways [6], such as uniformly. Ideally, given the set of patterns $\mathcal{P}$ to use in an LAD model, we could weight the patterns by solving the linear separation problem in the pattern space. Unfortunately, there

[1] LAD can also classify general numeric attributes and observations in R^n by generating attribute-value cut-points [6, 5].

are many more patterns than there are training observations, so it may not be tractable both to enumerate all patterns and to solve the linear separation problem by linear programming.

We propose to learn an LAD model from the classified observations Ω using *boosting*. Our contributions are:

- To propose a new branch-and-bound algorithm that finds the optimal maximum-agreement pattern. Section 2.1.1 presents our proposed algorithm and compares it to Dobkin's et al's algorithm.
- To adapt and implement the AdaBoost [13, 20] and SmoothBoost [21] algorithms in order to iteratively solve the separation problem in the space of logical pattern hypotheses. Section 2 describes our learning method, boosting, and the context where we suggest using our algorithm as a weak learner.
- To show that our exponential-time algorithm can learn logical patterns from real-world data. Section 3 compares the classification accuracy of our method to that of other state-of-the-art methods.

2 Boosting Logical Patterns

Boosting is an iterative process that combines a sequence of weak hypotheses h_t into an *ensemble hypothesis* of the form $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$. In each iteration t, boosting obtains the weak hypothesis h_t by invoking an unspecified *weak learner* with some weighted observations $d_t : \Omega \to [0, 1]$. For any subset O of Ω, we define $d_t(O) = \sum_{(x,y)\in O} d_t(x, y)$. The observation weights are normalized so that $d_t(\Omega) = 1$.

Algorithm 1 is Schapire and Singer's AdaBoost [20]. Boosting is guaranteed to converge to an ensemble that classifies all training observations correctly, as long as the weak hypotheses perform uniformly better than chance. More precisely, as long as the weak-hypothesis error ϵ_t is at most $1/2 - \gamma$ for some γ, the ensemble error is at most $\exp(-2\sum_{t=1}^{T} \gamma^2)$. Boosting also provides good separation margins.

2.1 Finding the Optimal Pattern

As one might expect, we apply boosting to weak hypotheses corresponding to positive and negative logical patterns, as defined in (1). In other words, each iteration of our weak learner chooses a pattern p to place in either $\mathcal{P}^+$ or $\mathcal{P}^-$. The original AdaBoost algorithm assumes that the weak hypotheses never abstain; that is, h_t maps each x to $+1$ or -1, never 0. In contrast, our weak hypotheses h_p may return 0.

One simple way to interpret the training error ϵ_t when the weak hypothesis h_t may abstain is to treat abstaining as flipping a fair coin to decide whether to return $+1$ or -1. This interpretation shows that we can still use the AdaBoost algorithm presented above. To do so, we want a weak learner that returns a weak hypothesis with as low training error as possible. The agnostic PAC-learning model [9, 17] considers such error-minimizing

Algorithm 1 AdaBoost [20]

Input: the observations Ω and the number of iterations T.

Initialize the observation weights $d_1(x, y) = 1/|\Omega|$.

for $t = 1, \ldots, T$ **do**

Generate the new weak hypothesis $h_t = \mathcal{L}(d_t)$.

Calculate the weighted training error

$$\epsilon_t = \sum\nolimits_{(x,y)\in\Omega} d_t(x, y)\frac{1 - yh_t(x)}{2}.$$

Set the hypothesis weight $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$.

Update the observation weights

$$d_{t+1}(x, y) = \frac{d_t(x, y)\exp\big(-\alpha_t y h_t(x)\big)}{Z_t},$$

where Z_t is a constant such that

$$\sum\nolimits_{(x,y)\in\Omega} d_{t+1}(x, y) = 1.$$

If $\epsilon_t = 0$ or $\epsilon_t \geq 1/2$, then set T to $t - 1$ and break out of the loop.

end for

Output: the ensemble hypothesis $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$.

hypotheses. Also, the LAD literature calls the homogeneous pattern with the lowest training error the *maximum* [12] or *strong* pattern [18]. Eckstein et al. proposed a branch-and-bound algorithm to solve a similar problem, to find the maximum box in R^n [11].

Given the weighted observations d_t, the training error of a pattern p is $\big(1 - |d_t^+(p) - d_t^-(p)|\big)/2$, where

$$\begin{aligned} d_t^+(p) &= d_t\big(\Omega \cap (\text{Cover}(p) \times \{+1\})\big), \\ d_t^-(p) &= d_t\big(\Omega \cap (\text{Cover}(p) \times \{-1\})\big) \end{aligned} \tag{3}$$

are respectively the total weights of positive and negative observations covered by p. Thus we want the pattern p to maximize

$$|d_t^+(p) - d_t^-(p)|. \tag{4}$$

We call the latter quantity the *score* of p. This problem is known as *monomial maximum agreement* in machine learning and *maximum bichromatic discrepancy* in computational geometry. This problem is NP-hard, as established by Kearns et al. [17], Ben-David et al. [2], and Feldman [12]. In fact, Feldman shows that it is NP-hard to approximate monomial maximum agreement to within any factor less than 2. Note that an approximation ratio of 2 is achieved by a trivial algorithm that assigns the majority value to the variable of any degree 1 monomial. Dobkin et al. give an exact solution for this problem by dynamic programming [9]. Their time complexity is $O(|\Omega|^{2n'-2} \log |\Omega|)$, where n' is the number of numeric attributes.

2.1.1 An Exact Search Algorithm

We solve monomial maximum agreement exactly using a branch-and-bound technique and a best-first-search queuing discipline to explore candidate patterns. The intuition behind our algorithm is that a pattern p's score $|d_t^+(p) - d_t^-(p)|$ never exceeds its *priority* $\max\big(d_t^+(p), d_t^-(p)\big)$. To find the pattern with the highest score, we apply best-first search to a directed tree of patterns, in which the children of a pattern p are patterns of the form $p \wedge x_i$ or $p \wedge \bar{x}_i$, where x_i is any attribute. In this tree, a child pattern never has higher priority than its parent. The root of the tree is the null pattern, which covers the entire hypercube and has the highest priority. Theorem 2.1 formalizes the correctness of our algorithm.

Algorithm 2 is our search algorithm. It explores the 3^n candidate patterns in non-ascending priority order. The best pattern is the best known pattern as soon as the best known score exceeds the priority of the candidate patterns left. In particular, because the score of a homogeneous pattern equals its priority, the algorithm stops right after it removes a homogeneous pattern from the queue.

Theorem 2.1. *MaximumAgreementPattern finds an optimal pattern that maximizes any score for which the priority* $\max\big(d_t^+(p), d_t^-(p)\big)$ *is an upper bound.*

Algorithm 2 MaximumAgreementPattern

Input: the weighted observations $d : \Omega \to [0, 1]$.

Initialize the best-known score $s^* \leftarrow 0$.
Initialize the best-known pattern $p^* \leftarrow$ none.
Initialize the priority queue of patterns Q to contain just the null pattern, which covers every attribute vector. The priority of a pattern p is $\max\bigl(d^+(p), d^-(p)\bigr)$.
while Q is not empty **do**
  Remove the top pattern p from the queue Q.
  Compute the pattern's score $s = |d^+(p) - d^-(p)|$.
  if $s > s^*$ **then**
    Record the new best-known score $s^* \leftarrow s$.
    Record the new best-known pattern $p^* \leftarrow p$.
  end if
  Break if the priority of p is no more than s^*.
  for $p' \in \{\, (p \wedge x_i),\ (p \wedge \bar{x}_i) \mid i = 1, \ldots, n \,\}$ **do**
    If the priority of p' is more than s^*, then add p' to Q.
  end for
end while

Output: the best pattern p^*.

Proof. Let p and p' denote the first two patterns in the queue Q. We claim the following loop invariant: either s^* or the priority $\max\bigl(d_t^+(p), d_t^-(p)\bigr)$ of p meets or exceeds the maximum score among all patterns, and p' has priority at most that of p.

At initialization this invariant clearly holds since p is the null pattern and no pattern's score or priority can be greater than the total weight of the observation class with greater total weight.

Each iteration maintains the invariant: If s^* is already maximum or if p has the maximum score and it is assigned to s^* then the invariant is maintained (the sequence of pattern scores assigned to s^* is strictly increasing). Otherwise, a maximum-score pattern appears in Q or descends from a pattern in Q (that is, extends a pattern in Q by conjoining literals). The next pattern p' in Q must have priority greater than the maximum score, because

1. the priority is an upper bound for the score, and

2. conjoining literals to a pattern never increases its priority.

By property 2, and since p' has the highest priority in Q, the priority of p is at least the priority of p'.

Upon termination the priority of p is less than or equal to s^*. Since the score of each pattern cannot exceed its priority, the invariant entails that s^* must be the maximum score. □

2.1.2 Complexity

Theorem 2.2. *The complexity of MaximumAgreementPattern is* $O(\min(3^n, 2^{|\Omega|})|\Omega|)$.

Proof. In the worst case (given a fixed Ω), 3^n logical patterns may need to be examined, each of which corresponds to an iteration of the main loop. Since we do not explore logical patterns that do not cover data points, we can bound the number of iterations by $\min(3^n, 2^{|\Omega|})$. In each iteration we compute the cover of each queued pattern. In order to do so we need to perform a single set-intersection operation[2] of complexity $O(|\Omega|)$. □

The complexity of Dobkin et al.'s algorithm is $O(|\Omega|^{2n'-2} \log |\Omega|)$. Our algorithm is preferable in the worst case to Dobkin et al.'s algorithm, when our count n of Boolean attributes is comparable to their count n' of real-valued attributes. The latter proviso holds:

1. either when the original attributes of the training data are already Boolean;

2. or when the original attributes are real-valued, and so need to be binarized for our learning algorithm (following Boros et al.'s procedure [5]), but the final count of binary attributes is small with respect to the number of data points and the original number of real-valued attributes. More specifically, the proviso holds when the final count of binary attributes n is less than $(2n' - 3) \log_3 |\Omega|$. Thus $n \in o(n' \log |\Omega|)$ is sufficient.

This is in fact the case for most real-life data: as seen in Figure 2, the number of binary variables required does not approach the upper bound for our datasets.

In practice, our algorithm runs fast enough thanks to pruning: the algorithm only bothers to queue a pattern p if its priority exceeds the current best-known score s^*. As the search gradually discovers better-scoring patterns p^*, this pruning threshold s^* improves and lets the algorithm ignore more patterns p. We further reduce the amount of time and space consumed by initializing s^* not to 0 as shown above but to the highest score among a few "promising" patterns, which is a better lower bound on the final best score. Our implementation gathers promising patterns from three heuristic sources:

1. every pattern in the current boosting ensemble;

2. a greedy search that starts at the null pattern and keeps choosing the child with the highest score;[3]

3. a best-first search with a fixed branching factor.

[2] This can be done by saving the set intersection so far for each logical pattern in the queue. In case of a depth-first queueing discipline this may only require linear space. In case of the best-first queuing discipline we choose, which performs better in practice (although with the same worst case complexity), the memory requirement could be exponential in theory but is moderate in practice.

[3] When the score is redefined to be $\left(\sqrt{d^+(p)} - \sqrt{d^-(p)}\right)^2$ as in (7), this heuristic corresponds to SLIPPER's weak learning algorithm [7].

To further improve performance, our algorithm can be easily adapted to find an approximate solution. For any predefined optimality gap g, the algorithm can ignore any candidate pattern whose priority does not exceed $(1+g)s^*$. For an optimality gap of g the algorithm is guaranteed to find a solution of at least the optimal score divided by $1+g$. Relaxing the optimality requirement by increasing g may allow scalability for large datasets. For example, for the UCI BUPA dataset [3], MaximumAgreementPattern terminates in about 15 seconds[4] with an optimal solution, given uniform observation weights. Relaxing the optimality only to a 5% gap reduces the running time by 15–20%.

2.1.3 Alternative Scoring

Because the priority of any pattern bounds its score from above, this algorithm always returns the pattern with the highest score. In fact, the same search strategy correctly yields the best-scoring pattern for any scoring function whose range is upper bounded by the priority. We discuss two such alternative scoring functions here.

First, following LAD tradition, we can require the pattern returned to be homogeneous or almost homogeneous. Define the *homogeneity* of a pattern p to be

$$\eta(p) = \frac{\max\big(d^+(p), d^-(p)\big)}{d^+(p) + d^-(p)} \in [1/2, 1]. \tag{5}$$

In the algorithm above, we can redefine the score s as

$$s = \begin{cases} |d^+(p) - d^-(p)| & \text{if } \eta(p) \geq \eta_0, \\ -\infty & \text{otherwise.} \end{cases} \tag{6}$$

Here η_0 is a global parameter for the learning algorithm, usually set to 1 or slightly less than 1.

The second alternative scoring function that is relevant to us is defined by

$$s = \big(\sqrt{d^+(p)} - \sqrt{d^-(p)}\big)^2. \tag{7}$$

It is useful as we turn in the next section to a variant of boosting designed for faster convergence of AdaBoost with abstaining weak hypotheses in mind.

An obstacle to maximizing both of these alternative scores is that they are lower than the original score in (4). The greater gap that results between the priority and score of patterns means that less pruning can take place. This drawback is striking when we compare our search algorithm with and without LAD's restriction on homogeneity: without the artificial restriction, the search actually considers more candidate patterns in less time and space. On the other hand, the second alternative scoring function, although more difficult to optimize within the weak learner, usually lets the AdaBoost algorithm converge faster, as explained in the sequel. We therefore use scoring function (7) for AdaBoost even though it makes the weak learner less computationally efficient.

[4] On a machine with an Intel i686 3.6 GHz CPU and 8960 MB RAM.

Dataset	Observations		Attributes	
	Positive	Negative	Original	Binarized
BCW	239	444	9	12
VOTE	268	167	16	11
BUPA	200	145	6	20
CLHEART	137	160	13	12
HUHEART	106	188	13	16

Table 1: Size and main characteristics of datasets used in the empirical study.

2.2 Boosting with Abstaining

Schapire and Singer introduce weak hypotheses that abstain in the context of boosting [20]. They propose to minimize the normalizing constant Z_t at each iteration. Because (as is easy to check)

$$Z = 1 - d^+(p) - d^-(p) + d^+(p)e^{\mp\alpha} + d^-(p)e^{\pm\alpha} \tag{8}$$

(where the signs in $\mp$ and $\pm$ depend on whether p is chosen as a positive or negative pattern), Schapire and Singer's proposal amounts to a more aggressive hypothesis weight:

$$\alpha = \pm\frac{1}{2}\ln\frac{d^+(p)}{d^-(p)}. \tag{9}$$

This choice of α gives $Z = 1 - \left(\sqrt{d^+(p)} - \sqrt{d^-(p)}\right)^2$ [7], so the weak learner should maximize (7).

In the case of a homogeneous logical pattern, however, either $d^+(p)$ or $d^-(p)$ is 0, so α in (9) is strictly speaking undefined. One way to deal with this case is to add a small smoothing constant to $d^+(p)$ and $d^-(p)$.

2.3 Handling Noisy Data with SmoothBoost

The AdaBoost algorithm increases the weights of observations that are hard to classify at each iteration. When the data is noisy, the increasing weight of those observations may overly emphasize outliers in the weak hypotheses and final ensemble.

To handle noisy data, boosting algorithms have been extended in several ways. First, boosting with *soft margins* "mistrusts" observations that influence the weak learner too much [18]. Alternatively, Servedio's SmoothBoost algorithm [21] explicitly constrains the distribution d_t of observation weights not to be too skewed. Algorithm 3 is SmoothBoost. It is not fully adaptive since it requires the user to define a desired error κ and margin θ. In our case we may disregard γ to be always satisfied since we use an optimizing weak learner (that is, we assume $\gamma = \theta$).

Algorithm 3 SmoothBoost [21]

Input: the observations Ω, the desired error κ, the required edge γ, and the desired margin θ, such that $0 < \kappa < 1$ and $0 \leq \theta \leq \gamma \leq 1/2$.

Initialize $M_1 : \Omega \to \mathbb{R}$ to $M_1(x, y) = 1$.
Initialize $N_0 : \Omega \to \mathbb{R}$ to $N_0(x, y) = 0$.
for $t = 1, \ldots$ **do**
Break if $|M_t|/|\Omega| < \kappa$.
Using the observation weights $d_t(x, y) = M_t(x, y)/|M_t|$, generate the new weak hypothesis $h_t = \mathcal{L}(d_t)$.
Update $N_t, M_{t+1} : \Omega \to \mathbb{R}$ to

$$N_t(x, y) = N_{t-1}(x, y) + yh_t(x) - \theta,$$

$$M_{t+1}(x, y) = \begin{cases} 1 & \text{if } N_t(x, y) < 0, \\ (1 - \gamma)^{N_t(x,y)/2} & \text{if } N_t(x, y) \geq 0. \end{cases}$$

end for

Output: the ensemble hypothesis $f(x) = \sum_t h_t(x)$.

3 Empirical Results

We test our algorithms Ada-BOP (which uses AdaBoost) and Smooth-BOP (which uses SmoothBooost) with MaximumAgreementPattern as a weak learner. We report results for 10-fold cross-validation, 5-fold cross-validation, or a fixed partition of the data into training and test sets, as indicated in Table 2. We compare these results to those reported for SLIPPER [7] and for LAD [6].

The LAD implementation [6] generates a large initial set of logical patterns up to a certain degree where each pattern must satisfy a desired level of homogeneity and coverage. It then solves a set covering problem to find a smaller set of patterns that covers the observations. It finally uses linear programming to compute the discriminant function that maximizes the margin of separation. These restrictions on the hypothesis space—in terms of pattern degree, pattern coverage, and homogeneity—are determined by both computational efficiency and test accuracy. Determining the exact parameter values to improve test accuracy can require extensive cross-validation. Moreover, we are unaware of any algorithm in the LAD literature for determining the best parameter values, so searching for these values may involve significant trial and error.

Table 1 shows the datasets that we use in our experimental study. We take our datasets from the UCI repository [3]. The datasets are the Wisconsin Breast Cancer (BCW), Congressional Voting (VOTE), Liver-disorders (BUPA), Cleveland Heart Disease (CLHEART), Hungarian Heart Disease (HUHEART), and Mushroom (MUSHR). We remove observations with missing attribute values, except in the VOTE dataset, where we treat missing values

as an additional attribute value "did not vote".[5]

To give a sense of the complexity of the data and the binarization step, the columns in Table 1 show the original and the binarized number of variables. The original number of variables may include both binary and real-valued variables. The number of binary variables is the number of variables in the support set obtained by the minimum set cover approximation separating each pair of training data points [6, 5]. In the actual test runs the number of binary variables used may be slightly less than the number indicated since less training data is used than the entire dataset.

The SmoothBoost algorithm requires the user to select the parameters of desired error rate κ and desired margin θ. How to set these parameters depends on how the observations and the noise are distributed in each specific problem. To capture these properties, we use a *smoothness* parameter [14] defined by

$$\beta = \sup_{(x,y)\in\Omega} \frac{d(x,y)}{D(x,y)}, \tag{10}$$

where $d(x,y)$ is the weight of the observation (x,y), and $D(x,y)$ is the probability of the data point (x,y) according to the target distribution. Examining the histogram of boosting data weights, we find that most data points are concentrated around the uniform weighting with an additional peak of significantly heavier weights. This is evident in Figure 1 even after 400 boosting iterations; such behavior is typical for most datasets we encounter. Accordingly we assume that the target distribution is uniform across observations (that is, $D(x,y)$ equal for all (x,y) in Ω). The smoothness parameter can be used to derive an upper bound on the weight of data points. Imposing a limit on the data weights in boosting deemphasizes the effect of outliers in determining the weights (or separating hyperplane) of the hypotheses. The final data weights of AdaBoost along with our fixed smoothness parameter can be used to determine the outliers in the data as those points which take on values much larger than the uniform weight value. In particular, in our example (Figure 1) of training data with 267 observations, uniform weights $D(x,y)$ should take on the value $1/267 \approx 0.004$. Requiring a smoothness value of 3.75 implies that no observation should get a weight greater than 0.014. The value of 3.75 can be justified in Figure 1 by the fact that the distribution tail and additional peak of the distribution centered at 0.015 shall be considered as outliers. Consequently imposing the limit on weight would alleviate the skewness of the distribution depicted in the diagram. We adopt this value for all datasets so that we enforce $\beta \leq 3.75$. Finally we set κ and θ in SmoothBoost as follows: We first run AdaBoost and treat those observations whose weights end up greater than $3.75/|\Omega|$ as outliers. Then, we set the desired error rate κ to the proportion of observations that are outliers. We also set the desired margin θ to the minimum margin that AdaBoost attained on the non-outlier observations.

Table 2 shows the preliminary results of our experimental study. First we note that the Smooth-BOP provides a substantial improvement in generalization over Ada-BOP. This may

[5]The LAD implementation we compare against [6] handles missing attribute values in a more sophisticated way.

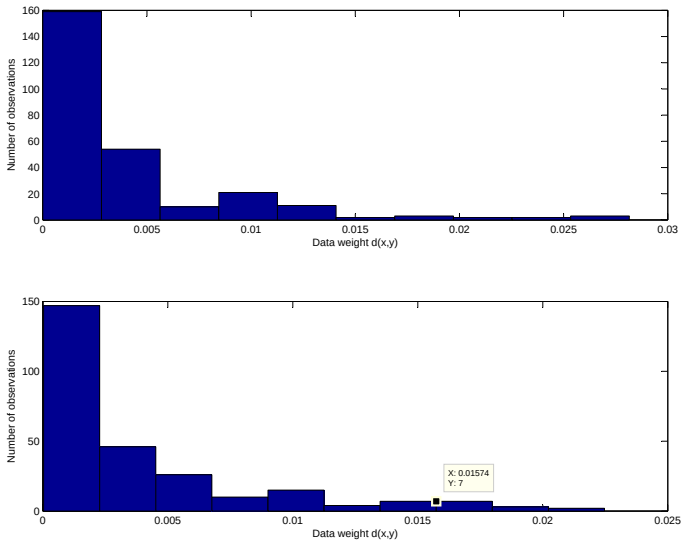

Figure 1: Histogram of CLHEART dataset boosting data weights after 50 (top) and 400 (bottom) iterations

serve as evidence to the fact that AdaBoost is sensitive to the existence of outliers and noise (it is important to note that we allow weak hypotheses that could overfit e.g. monomials of high degree). Particularly, Ada-BOP performs even more poorly on datasets for which the best known classification accuracies tend to be lower, such as CLHEART and BUPA. The low classification accuracy of many different classification methods may suggest that these datasets are noisy. Also important, the results suggest that our algorithm (Smooth-BOP) outperforms SLIPPER on all of the datasets[6]. LAD outperforms Smooth-BOP and SLIPPER. The relatively small advantage of LAD in terms of classification accuracy comes at the expense of greater computational effort involved in enumeration of all patterns and linear programming as well as extensive fine tuning in order to adjust the free parameters, as discussed in the above.

4 Conclusion

Our branch-and-bound (best-first search) algorithm solves the maximum agreement problem in a hypercube of arbitrary dimension. Used as the weak learner in boosting, the algorithm performs well in practice despite the worst-case exponential time and space complexity.

When boosting our optimizing weak learner, SmoothBoost improves significantly over

[6]Although this claim cannot be supported at a high level of confidence we expect that running additional k-fold experiments shall increase the level of confidence of this claim with respect to the variability of the classification accuracy of our method. We use the accuracies for the other methods as reported in the literature. In the case of SLIPPER no standard deviation is reported.

Dataset	Method	Accuracy on test set					
		LAD	SLIPPER	Ada-BOP		Smooth-BOP	
		Average	Average	Average	S. Dev.	Average	S. Dev.
BCW	10CV	0.972	0.958	0.944	0.025	0.963	0.019
VOTE	10CV	0.966		0.943	0.015	0.957	0.027
	135 test		0.970	0.956	(NA)	0.970	(NA)
BUPA	10CV	0.723	0.678	0.622	0.063	0.684	0.046
CLHEART	5CV	0.838		0.752	0.049	0.813	0.045
HUHEART	10CV		0.806	0.751	0.085	0.791	0.081
MUSHR	Other[1]	0.981[2]	0.998	0.999	0.001	0.998	0.001

Table 2: Experiment datasets and results. The LAD accuracies reported of Boros et al. [6] are not obtained through cross validation. [1]SLIPPER uses a fixed training set of slightly less than 50% of the data, LAD results [1] are based on 2 fold cross validation. We use random sampling to create smaller training sets consisting of about 17% of the data. [2]The results [1] reported for this dataset average the performance of LAD with 6 other machine learning algorithms.

AdaBoost in the face of noisy data. The combination of SmoothBoost and our weak learner outperforms other state-of-the-art learning algorithms that combine abstaining logical patterns, such as SLIPPER. Our algorithm is fairly competitive with optimizing learning algorithms such as LAD. Nevertheless, the only free parameter in our method is the number of boosting iterations, an advantage over the several free parameters in LAD.

Finally, we give additional insight by suggesting that any boosting algorithm can be used with our proposed weak learner to generate LAD models more efficiently. For example, using LP boosting [10] along with our weak learner, would correspond to a column generation approach to solving the LAD margin maximization linear program. In any case we propose that using our weak learner one can construct an optimal ensemble of logical patterns without having to enumerate all, and with good run time performance in practice, as tested on a number UCI datasets [3].

References

[1] Anthony M., Hammer P. L., Subasi E., Subasi M. *Using a similarity measure for credible classification*. RUTCOR Research Report 39-2005, 2005.

[2] Ben-David S., Eiron N., Long P. M. *On the Difficulty of Approximately Maximizing Agreements*. Journal of Computer and System Sciences Vol. 66(3), 2003.

[3] Blake C. L., Hettich S., Merz C. J., Newman, D. J. *UCI Repository of machine learning databases*. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, University of California, Department of Information and Computer Science, Irvine, CA, 1998.

[4] Bonates T. O., Hammer P. L., Kogan A. *Maximum Patterns in Datasets*. RUTCOR Research Report 9-2006, 2006.

[5] Boros E., P.L. Hammer, T. Ibaraki, A. Kogan. *Logical Analysis of Numerical Data*. Mathematical Programming, 79, 1997, pp. 163–190.

[6] Boros E., Hammer P. L., Ibarki T., Kogan A., Mayoraz E., Muchnik I. *An Implementation of Logical Analysis of Data*. IEEE Transactions on Knowledge and Data Engineering, Vol. 12 (2), 2000, pp. 292–306.

[7] Cohen W. W., Singer Y. *A Simple, Fast, and Effective Rule Learner*. In Proceedings of the Sixteenth National Conference on Artificial Intelligence, Menlo Park, CA, 1999, AAAI/MIT Press, pp. 335–342.

[8] Crama Y., Hammer P. L., Ibarki T. *Cause Effect Relationships and partially defined Boolean functions*. Annals of Operations Operations Research, 16, 1988, pp. 299–326.

[9] Dobkin D. P., Gunopulos D., Maas W. *Computing the Maximum Bichromatic Discrepancy, with Applications to Computer Graphics and Machine Learning*. Journal of Computer and System Sciences, 52(3), 1996, pp. 453–470.

[10] Demiriz A., Bennett K. P., Shawe-Taylor, J. *Linear Programming Boosting via Column Generation*. Machine Learning, 46(1), 2001, pp. 225–254.

[11] Eckstein J., Hammer P. L., Liu Y., Nediak M., Simeone B. *The Maximum Box Problem and its Application to Data Analysis*. Computational Optimization and Applications, 23, 2002, pp 285–298.

[12] Feldman V. *Optimal Hardness Results for Maximizing Agreements with Monomials*. 21st Annual IEEE Conference on Computational Complexity (CCC'06), 2006, pp. 226–236.

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

[14] Gavinsky D. *Optimally-smooth adaptive boosting and application to agnostic learning*. Journal of Machine Learning Research, 4, 2003, pp. 101–117.

[15] Hammer P. L., Kogan A., Simeone B., Szedmark S. *Pareto-optimal patterns in logical analysis of data*. Discrete Applied Mathematics 144(1–2), 2004, pp. 79–102.

[16] Kearns M., Li M. *Learning in the presence of malicious errors*. SIAM Journal on Computing. Vol 22(4), 1993, pp. 807–837.

[17] Kearns M. J., Schapire R. E., Sellie L. M. *Toward Efficient Agnostic Learning*. Machine Learning 17, 1994, pp. 115-141.

[18] Ratsch G., Onoda T., Muller K. R. *Soft Margins for AdaBoost*. Machine Learning 42(3), 2001, pp. 287–320.

[19] Ratsch G., Warmuth M. K., Mika S., Onoda T., Lemm S., Muller K. R. *Barrier Boosting*. COLT 2000, pp. 170–179.

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

[21] Servedio R. A. *Smooth Boosting and Learning with Malicious Noise*. Journal of Machine Learning Research 4, 2003, pp. 633–648.

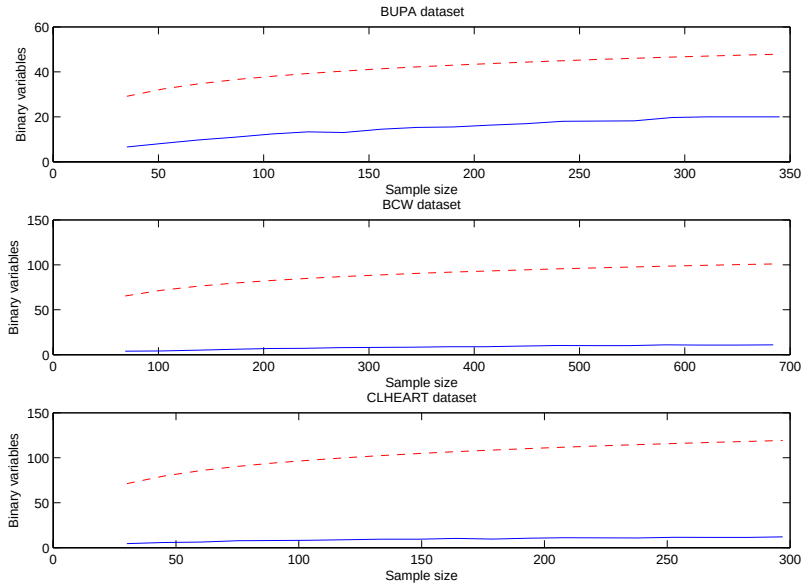

Figure 2: The number of binary variables required as a function of the sample size used for the BUPA (top), BCW (middle) and CLHEART (bottom) datasets. The solid line is the actual number of binary variables, whereas the dotted line is the upper bound.