An Improved Branch-and-Bound Method for Maximum Monomial Agreement

Jonathan Eckstein
MSIS Department and RUTCOR
Rutgers University
Piscataway, NJ 08854
jeckstei@rci.rutgers.edu

Noam Goldberg
RUTCOR
Rutgers University
Piscataway, NJ 08854
ngoldberg@rutcor.rutgers.edu

Abstract

The $\mathcal{NP}$-hard *maximum monomial agreement* (MMA) problem consists of finding a single logical conjunction that best fits a weighted dataset of "positive" and "negative" binary vectors. Computing classifiers using boosting methods involves a maximum agreement subproblem at each iteration, although such subproblems are typically solved by heuristic methods. Here, we describe an exact branch and bound method for maximum agreement over Boolean monomials, improving on the earlier work of Goldberg and Shan [14]. In particular, we develop a tighter upper bounding function and an improved branching procedure that exploits knowledge of the bound and the dataset, while having a lower branching factor. Experimental results show that the new method is able to solve larger problem instances and runs faster within a linear programming boosting procedure applied to medium-sized datasets from the UCI repository.

1 Problem Statement and Introduction

The *maximum monomial agreement* (MMA) problem has applications to machine learning, data mining [15, 14], as well as computational geometry and computer graphics [8], where it is known as the *maximum bi-chromatic discrepancy* problem. As input, we will assume a given set of m binary n-vectors in the form of a 0/1 matrix $A \in \{0,1\}^{m\times n}$, along with a partition of its rows into positive observations $\Omega^+ \subset \{1,...,m\}$ and negative observations $\Omega^- = \{1,...,m\} \setminus \Omega^+$. We are also given a function $w : \{1,\ldots,m\} \to [0,\infty)$; $w(i)$ specifies the weight of observation i. A monomial on $\{0,1\}^n$ is simply a function $p : \{0,1\}^n \to \{0,1\}$ of the form

$$p_{J,C}(x) = \prod_{j\in J} x_j \prod_{c\in C}(1-x_c), \tag{1}$$

where J and C are disjoint subsets of $\{1,\ldots,n\}$. The monomial $p_{J,C}$ is said to *cover* a vector $x \in \{0,1\}^n$ if $p_{J,C}(x) = 1$. We define the *coverage* of a monomial $p_{J,C}$ to be

$$\mathrm{Cover}_A(J,C) = \{i \in \{1,...,m\} \mid p_{J,C}(A_i) = 1\},$$

where A_i denotes the i^{th} row of A. Defining the weight of a set $S \subseteq \{1,\ldots,m\}$ to be $w(S) = \sum_{i\in S} w(i)$, the MMA problem is to find disjoint sets $J, C \subseteq \{1,\ldots,n\}$ maximizing

$$f(J,C) = \left|w(\mathrm{Cover}(J,C)\cap\Omega^+) - w(\mathrm{Cover}(J,C)\cap\Omega^-)\right|. \tag{2}$$

When the problem dimension n is part of the input, this problem is known to be $\mathcal{NP}$-hard [15]. Furthermore, it has been shown [11] that even if the weights $w(i)$ are all equal, a related problem with a slightly different objective function is $\mathcal{NP}$-hard to approximate to any factor less than 2.

In this setting, we consider only binary datasets; however, for general datasets in $\mathbb{R}^n$, we note that there is a corresponding "binarization" with dimension that is at most polynomially larger than m,

and in practice does not tend to be much larger than n [3, 14]. A related (but not equivalent) problem for real-valued data is the *maximum box* problem [9].

We are particularly interested in applications of the MMA problem to supervised learning and classification. For example, we may have m patients, with Ω^+ corresponding to those having a particular disease, and Ω^- to those who do not. Each patient has n binary attributes corresponding to personal traits or results of medical tests, and we would like to find the interaction of attributes which best predicts presence or absence of the disease. For example, a monomial might represent a rule or hypothesis such as ('age' ≥ 50) $\wedge$ ('blood pressure' = 'high') $\rightarrow$ 'heart disease'.

In classification problems, monomial hypotheses have been found especially useful when used in ensembles, typically by thresholding linear combinations of monomials. In the machine learning community, iterative algorithms that linearly combine classifiers into a "stronger" ensemble classifier originated with Schapire and Freund [12] and are referred to as *boosting algorithms*. In particular, boosting of monomial hypotheses has been suggested by several authors [6, 13, 14]. The results in [14] show that boosting optimal monomial hypotheses, as opposed to heuristically generated monomials (*e.g.* as in SLIPPER [6]), can improve the classification performance when using a sufficiently robust boosting algorithm, such as Servedio's SmoothBoost [17]. Demiriz, Bennett and Shawe-Taylor [7] suggest a boosting method for general weak learner hypotheses (not necessarily monomials), based on column generation linear programming techniques, using a robust reformulation of an LP model for finding a separating hyperplane; see also [16]. In their column generation subproblem, the objective is precisely that of maximum agreement; if the space of possible weak learners consists of all monomials, the subproblem is precisely to maximize (2). Monomial hypotheses (also called logical patterns) are also a basic building block in the *logical analysis of data* (LAD) methodology [4], where linear programming techniques are also used to compute the discriminant function.

2 Branch and bound

One possible approach to exactly solving the MMA is to formulate it as an equivalent integer linear program, and solve it with a standard integer programming solver. However, we have found it is more efficient to solve the problem by a specialized branch-and-bound algorithm. The key elements of a branch-and-bound algorithm are the definition of subproblems that represent sets of possible solutions, a method for computing a bound on the objective value of all solutions represented by a subproblem, and a branching procedure for subdividing subproblems into smaller ones.

We characterize each subproblem as a partition (J, C, E, F) of $\{1, \ldots, n\}$. As in (1), J and C respectively represent the literals which must be in the monomial, and whose complements must be in the monomial. E indicates a set of "excluded" literals: neither they nor their complements may appear in the monomial. Finally, $F = \{1, \ldots, n\} \backslash (J \cup C \cup E)$ is the set of "free", undetermined literals. The set of monomials corresponding to subproblem (J, C, E, F) is given by

$$P(J, C, E) = \{(J', C') \mid J' \supseteq J,\ C' \supseteq C,\ \text{and } J', C', E \text{ are disjoint}\}. \tag{3}$$

The branch-and-bound search begins with a priority queue Q containing the single root subproblem $(\emptyset, \emptyset, \emptyset, \{1 \ldots, n\})$, which corresponds to the set of all possible monomials. At each iteration, we remove a subproblem (J, C, E, F) from Q and, unless it is fathomed, that is, its upper bound $u(J, C, E) \leq f(J^*, C^*)$ for some known J^*, C^*, we further subdivide (branch) it into smaller subproblems. For the fathoming test to be valid, the upper bound function u should have the property

$$u(J, C, E) \geq f(J', C') \quad \forall\, (J', C') \in P(J, C, E). \tag{4}$$

2.1 The upper bound function

In the special case that $F = \emptyset$, the set $P(J, C, E)$ is a singleton consisting only of (J, C), and we can take $u(J, C, E) = f(J, C)$. If $F \neq \emptyset$, we must use some other function satisfying (4). Goldberg and Shan [14] suggest the simple bound $u(J, C, E) = u_{\text{gs}}(J, C)$, where

$$u_{\text{gs}}(J, C) = \max\{w(\text{Cover}(J, C) \cap \Omega^+), w(\text{Cover}(J, C) \cap \Omega^-)\} \tag{5}$$

It is straightforward to establish that this bound satisfies (4). However, it essentially ignores the information in the set E. To obtain a stronger bound, we now exploit the information in E:

Definition 2.1. *Two binary vectors $x, y \in \{0,1\}^n$ are* inseparable *with respect to a set $E \subseteq \{1,...,n\}$, if, for all $j \in \{1,...,n\} \setminus E$, one has $x_j = y_j$.*

Inseparability is an equivalence relation: any set $E \subseteq \{1,\ldots,n\}$ thus partitions $\{1,...,m\}$ into equivalence classes, i and i' being in the same class when the observation A_i is inseparable from the observation $A_{i'}$. Let us denote these classes by $V_1^E, V_2^E, ..., V_{q(E)}^E$, where $1 \leq q(E) \leq m$. A similar technique has been used by De Bontridder *et al.* [2] in the context of the minimum test cover problem. Let

$$w_\ell^+(J,C,E) = w(V_\ell^E \cap \mathrm{Cover}(J,C) \cap \Omega^+) \qquad w^+(J,C) = w(\mathrm{Cover}(J,C) \cap \Omega^+)$$
$$w_\ell^-(J,C,E) = w(V_\ell^E \cap \mathrm{Cover}(J,C) \cap \Omega^-) \qquad w^-(J,C) = w(\mathrm{Cover}(J,C) \cap \Omega^-)$$

for $\ell = 1,\ldots,q(E)$, and note that

$$f(J,C) = \left|w^+(J,C) - w^-(J,C)\right| = \left|\sum_{\ell=1}^{q(E)} (w_\ell^+(J,C,E) - w_\ell^-(J,C,E))\right|. \tag{6}$$

We will call a monomial *positive* if $w^+(J,C) \geq w^-(J,C)$, and *negative* if $w^+(J,C) < w^-(J,C)$. Positive monomials cover more weight of positive than negative observations, and negative monomials the reverse; we include ties in the positive class.

Now consider some $(J',C') \in P(J,C,E)$, assume it is positive, and fix some $\ell \in \{1,\ldots,q(E)\}$. Because the observations in the equivalence class V_ℓ^E are inseparable with respect to E, the monomial $p_{J',C'}(x)$ must either cover all of V_ℓ^E, or cover none of it. Considering the ℓ^{th} term in the second expression in (6), we see that the largest value of $f(J',C')$ would result if $p_{J',C'}(x)$ covers the entire class V_ℓ^E whenever $w_\ell^+(J,C,E) \geq w_\ell^-(J,C,E)$, obtaining a value of $w_\ell^+(J,C,E) - w_\ell^-(J,C,E)$, and otherwise does not cover the the entire class V_ℓ^E, obtaining the value 0. Thus, if (J',C') is positive, we have

$$f(J',C') \leq \sum_{\ell=1}^{q(E)} \left(w_\ell^+(J,C,E) - w_\ell^-(J,C,E)\right)_+,$$

where $(\cdot)_+$ denotes the positive part of a number. We can demonstrate a similar relation for the case that (J',C') is negative, and combining the two cases, we derive the following upper bound function satisfying (4):

$$u(J,C,E) = \max\left\{\begin{array}{l} \sum_{\ell=1}^{q(E)} \left(w_\ell^+(J,C,E) - w_\ell^-(J,C,E)\right)_+, \\ \sum_{\ell=1}^{q(E)} \left(w_\ell^-(J,C,E) - w_\ell^+(J,C,E)\right)_+ \end{array}\right\} \tag{7}$$

$$= u_{\mathrm{gs}}(J,C) - \sum_{\ell=1}^{q(E)} \min\{w_\ell^+(J,C,E), w_\ell^-(J,C,E)\} \tag{8}$$

For brevity, we omit the (fairly straightforward) proof that (7) and (8) are equivalent. We define $\phi(J,C,E) = \sum_{\ell=1}^{q(E)} \min\{w_\ell^+(J,C,E), w_\ell^-(J,C,E)\}$, the second term in (8), to be the *inseparability* of E with respect to J and C. Unless $\phi(J,C,E) = 0$, the bound $u(J,C,E)$ is strictly tighter than $u_{\mathrm{gs}}(J,C)$. If $\phi(J,C,E) = 0$, then all the sets $\mathrm{Cover}(J,C) \cap V_\ell^E$, for $\ell = 1,\ldots,q(E)$, are *homogeneous*, meaning that they contain only positive or only negative observations (with positive weights).

Both of the bounds $u_{\mathrm{gs}}(R,C)$ and (7) may be computed in $O(mn)$ time in the worst case (although we omit the details here). However, computing the equivalence classes $\{V_\ell^E\}$ in $O(mn)$ time requires a radix sort, which is $\Theta(mn)$. Computing $u_{\mathrm{gs}}(R,C)$ involves n set intersection operations which tend to be much faster than $O(m)$ in practice. Thus, it may be more practically efficient to first compute $u_{\mathrm{gs}}(R,C)$, and if that does not fathom the subproblem (J,C,E,F), then compute $\phi(J,C,E)$ and thus (7).

2.2 The branching procedure

Our branching procedure works as follows: given a subproblem (J,C,E,F), we select k distinct elements $j_1,\ldots,j_k$ of F, where $1 \leq k \leq |F|$. In our branching step, we create $2k+1$ smaller subproblems respectively representing the following subsets of $P(J,C,E)$:

- The subset of monomials in which none of $x_{j_1}, \ldots, x_{j_k}$ appear.
- For $t = 1, \ldots, k$ the subset of monomials in which x_{j_t} is the first in the sequence $x_{j_1}, \ldots, x_{j_k}$ to appear, uncomplemented; $x_{j_1}, \ldots, x_{j_{t-1}}$ are excluded from further consideration.
- For $t = 1, \ldots, k$ the subset of monomials in which x_{j_t} is the first in the sequence $x_{j_1}, \ldots, x_{j_k}$ to appear, and is complemented; $x_{j_1}, \ldots, x_{j_{t-1}}$ are excluded from further consideration.

These subsets clearly form a partition of $P(J, C, E)$. In our notation, the corresponding subproblems are represented by the respective 4-tuples in lines 13, 15, and 16 of Algorithm 1 below. We have significant latitude in choosing k for each subproblem. At present, we simply choose some $\alpha \in (0, 1]$, and set $k = \lceil \alpha |F| \rceil$ for each subproblem; we have also experimented with fixing $k = 1$.

Motivated by (8), we attempt for a given k to choose $j_1, \ldots, j_k$ to maximize the inseparability $\phi(J, C, E \cup \{j_1, \ldots, j_k\})$. Although we omit the details here, we have proved that exactly maximizing $\phi(J, C, E \cup \{j_1, \ldots, j_k\})$ is itself $\mathcal{NP}$-hard. However, we are also able to show that $\phi(J, C, E)$ is a supermodular function of E, motivating the use of a reverse greedy heuristic (or "stingy" algorithm [5]). Conceptually, we set $E' \leftarrow F$, and then remove elements one-by-one from E', greedily with respect to maximizing $\phi(J, C, E \cup E')$, until $|E'| = k$. We then take $\{j_1, \ldots, j_k\} = E'$, and determine the ordering $j_1, \ldots, j_k$ by a similar reverse greedy procedure.

We now consider the case $k = 1$, in which case our branching scheme generates three children. In this case, we implement an alternative, "strong" branching method: for each possible branching feature $j \in F$, we calculate the bounds $u(J \cup \{j\}, C, E)$, $u(J, C \cup \{j\}, E)$, and $u(J, C, E \cup \{j\})$ of the three resulting subproblems. For each j, we place these bounds in a triple $\mathbf{u}_j$ with elements sorted in descending order, and branch on some feature j that lexically minimizes $\mathbf{u}_j$. This approach is not only faster than the reverse greedy method, but also more practically effective at pruning search nodes. One reason for the efficiency of this alternative branching strategy is that it considers the bounds of all three prospective children, and not just the bound of the child $(J, C, E \cup \{j\}, F \setminus \{j\})$. Algorithm 1 outlines our entire branch-and-bound algorithm; there, our procedure for choosing $j_1, \ldots, j_k$ is called *exclude*.

Algorithm 1 MaxMonom

```
 1: Input: Sets Ω+, Ω−, and m × n matrix A.
 2: Output: Best solution value l and corresponding monomial given by (J*, C*).
 3: Q ← {(∅, ∅, ∅, {1 ..., n})}
 4: l ← −∞
 5: while Q ≠ ∅ do
 6:    Remove subproblem S = (J, C, E, F) from Q
 7:    if u(J, C, E) > l then
 8:       if f(J, C) > l then
 9:          (J*, C*) ← (J, C)
10:          l ← f(J*, C*)
11:       end if
12:       (j_1, ..., j_k) ← exclude(J, C, F)
13:       Insert (J, C, E ∪ {j_1, ..., j_k}, F \ {j_1, ..., j_k}) into Q
14:       for t ∈ {1, ..., k} do
15:          Insert (J ∪ {j_t}, C, E ∪ {j_1, ..., j_{t−1}}, F \ {j_1, ..., j_t}) into Q
16:          Insert (J, C ∪ {j_t}, E ∪ {j_1, ..., j_{t−1}}, F \ {j_1, ..., j_t}) into Q
17:       end for
18:    end if
19: end while
```

We note that in the earlier branch-and-bound method of [14], the branching procedure is essentially the special case that k is always chosen as large as possible, that is, $k = |F|$, and $j_1, \ldots, j_k$ are ordered so that $j_1 < j_2 < \cdots < j_k$. In this case, the $(J, C, E \cup \{j_1, \ldots, j_k\}, F \setminus \{j_1, \ldots, j_k\})$ child always represents exactly one monomial, allowing it to be immediately evaluated and implicitly dropped from the search tree.

In lines 15 and 16 of Algorithm 1, we apply a few simple pruning rules that significantly improve practical performance: first, we need not insert a subproblem into Q if it is already fath-

omed. Second, it can be shown that if $\mathrm{Cover}(J \cup \{j_t\}, C) = \mathrm{Cover}(J, C)$, the subproblem $(J \cup \{j_t\}, C, E \cup \{j_1, \ldots, j_{t-1}\}, F \setminus \{j_1, \ldots, j_t\})$ cannot contain a solution strictly better than those in $(J, C, E \cup \{j_1, \ldots, j_t\}, F \setminus \{j_1, \ldots, j_t\})$. Thus, such a subproblem may be dropped from consideration without insertion into Q. Finally, a similar result holds for the subproblem $(J, C \cup \{j_t\}, E \cup \{j_1, \ldots, j_{t-1}\}, F \setminus \{j_1, \ldots, j_t\})$ when $\mathrm{Cover}(J, C \cup \{j_t\}) = \mathrm{Cover}(J, C)$.

3 Experimental study

We implemented the procedure of Algorithm 1 in C++ using the the open-source PEBBL branch-and-bound library [10]. We ran our experiments on a workstation with 3.00 GHz Intel Xeon processors and 800MHz memory (running only in serial, although PEBBL is capable of parallelism).

Table 1 shows our experimental results using Algorithm 1 as a weak learning algorithm inside the LP-Boost boosting procedure [7], applied to the the UCI [1] binary dataset SPECTHRT and binarized versions of additional UCI datasets. We configured our binarization procedure to obtain a larger number of variables (as indicated by n in the table) than is customary for most binary feature selection methods (*e.g.* [3]).

The LP-Boost algorithm requires a "soft margin" penalty parameter D. We selected $D = 3/m$, which in our computational experience provides good classification and generalization performance. We compare four different algorithm configurations:

- An algorithm equivalent to [14], in which $k = |F|$, using only the u_{gs} upper bound function
- Algorithm 1, with $k = |F|$ and the bound (7)
- Algorithm 1, with $k = \lceil |F| /2 \rceil$ and the bound (7).
- Algorithm 1, with $k = 1$, the bound (7), and the "strong" $k = 1$ branching rule described above. Note that $k = 1$ corresponds to a ternary search tree.

We used a best-first queueing discipline with the queue size limited to at most 500,000 subproblems. We also limited the CPU time of each branch-and-bound search to 30 minutes. LP-Boost starts with the weights $w(i)$ equal for all i, but subsequently adjusts the observation weights based on the dual variables of its LP formulation's separation constraints. We ran each case for 30 iterations, or until a subproblem encountered the queue size or branch-and-bound time limit. As is well-known in practice, boosting algorithms tend to focus their later iterations on observations that are more difficult to classify. In our case, the later iterations produce longer monomials whose $|\mathrm{Cover}(J, C)|$ is smaller. The later iterations also tend to have larger search trees and longer running times.

Two conclusions appear to follow from the results in Table 1. First, compared with [14], both our tighter bound and our new reverse greedy branching scheme significantly decrease the number of search nodes required; they allow larger problems to be solved, and improve running time in all but the easiest cases involving SPECTHRT. Second, choosing an intermediate number of branching features $k = \lceil |F| /2 \rceil$ performs better in terms of search nodes than the two extreme strategies $k = |F|$ and $k = 1$; although space does not permit us to report full results, we also found that $k = \lceil |F| /2 \rceil$ yields smaller search trees than $k = \lceil 3 |F| /4 \rceil$ and $k = \lceil |F| /4 \rceil$. Finally, although taking $k = 1$ is not the best strategy in terms of search tree size, its specialized, faster branching procedure performs well enough that $k = 1$ is clearly best in terms of runtime. Thus, our new bound coupled with the $k = 1$ ternary branching scheme significantly outperform the method of [14] for larger datasets, and in later boosting iterations, when the weights become more "difficult".

Acknowledgements We would like to thank Chung-chieh Shan and Endre Boros for their contributions to our development process.

References

[1] Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007.

[2] Koen M. J. De Bontridder, B. J. Lageweg, Jan K. Lenstra, James B. Orlin, and Leen Stougie. Branch-and-bound algorithms for the test cover problem. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 223–233, London, UK, 2002. Springer-Verlag.

[3] Endre Boros, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan. Logical analysis of numerical data. *Mathematical Programming*, 79:163–190, 1997.

Dataset # Variables	LP-Boost Iters	$u_{\rm gs}$ bound $k = \lvert F\rvert$ CPU Sec	$u_{\rm gs}$ bound $k = \lvert F\rvert$ BB Nodes	$k = \lvert F\rvert$ CPU Sec	$k = \lvert F\rvert$ BB Nodes	$k = \lceil \lvert F\rvert /2\rceil$ CPU Sec	$k = \lceil \lvert F\rvert /2\rceil$ BB Nodes	$k = 1$ CPU Sec	$k = 1$ BB Nodes
SPECTHRT	1-15	0.4	7791.5	0.8	2537.1	0.6	1488.3	0.1	50.5
$n = 22$	16-30	1.15	22751.1	2.1	13080.9	1.6	6110.9	0.3	162.9
CLEVHRT	1-15	22.1	89551.2	18.2	1403.7	17.5	663.5	9.8	1638.3
$n = 35$	16-30	71.5	317537.9	44.3	4846.4	41.6	2014.5	24.4	4831.6
HEPATITIS	1-15	15.8	85060.1	7.2	644.1	7.1	479.8	3.2	852.6
$n = 37$	16-30		Q LIMIT	10.9	1132.2	10.8	763.5	5.9	1744.5
PIMA	1-15		Q LIMIT	100.3	3050.5	92.9	1435.7	67.6	4606.3
$n = 33$	16-30		Q LIMIT	327.2	19312.5	285.9	6266.5	235.9	21709.5
CMC	1-10		Q LIMIT	LIMIT		LIMIT		249.8	1753.7
$n = 58$	11-20		Q LIMIT	LIMIT		LIMIT		604.3	4779.8
HUNGHRT	1-15		Q LIMIT	LIMIT		LIMIT		269.0	14169.3
$n = 72$	16-30		Q LIMIT	LIMIT		LIMIT		784.8	47657.2

Table 1: Runtime and node averages over the specified LP-Boost iterations, applying Algorithm 1 to the indicated (binarized) UCI [1] datasets. "Q LIMIT" indicates an iteration encountered the 500,000-node queue limit, and "LIMIT" indicates an iteration encountered the 30-minute time limit.

[4] Endre Boros, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, Eddy Mayoraz, and Ilya Muchnik. An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):292–306, 2000.

[5] Endre Boros, Takashi Horiyama, Toshihide Ibaraki, Kazuhisa Makino, and Mutsunori Yagiura. Finding essential attributes from binary data. *Annals of Mathematics and Artificial Intelligence*, 39(3):223–257, 2003.

[6] William W. Cohen and Yoram Singer. A simple, fast, and effective rule learner. In *In Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 335–342, 1999.

[7] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46:225–254, 2002.

[8] David P. Dobkin, Dimitrios Gunopulos, and Wolfgang Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *Journal of Computer and Systems Sciences*, 52(3):453–470, 1996.

[9] Jonathan Eckstein, Peter L. Hammer, Ying Liu, Mikhail Nediak, and Bruno Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002.

[10] Jonathan Eckstein, Cynthia A. Phillips, and William E. Hart. PEBBL 1.0 user guide. RUTCOR Research Report RRR 19-2006, RUTCOR, Rutgers University, 2006.

[11] Vitaly Feldman. Optimal hardness results for maximizing agreements with monomials. In *Proceedings of the Annual IEEE Conference on Computational Complexity*, pages 226–236, 2006.

[12] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and Systems Sciences*, 55(1):119–139, 1997.

[13] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *Annals of Applied Statistics*, 2(3):916–954, 2008.

[14] Noam Goldberg and Chung-chieh Shan. Boosting optimal logical patterns. In Chid Apte, Bing Liu, Srinivasan Parthasarathy, and David Skillicorn, editors, *Proceedings of the Seventh SIAM International Conference on Data Mining*, 2007.

[15] Michael J. Kearns, Robert E. Schapire, and Linda M. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3):115–141, 1994.

[16] Gunnar Rätsch, Bernhard Schölkopf, Alex J. Smola, Sebastian Mika, Takashi Onoda, and Klaus-Robert Müller. Robust ensemble learning. In Alex J. Smola, Peter J. Bartlett, Bernhard Schölkopf, and Dale Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 207–219. MIT Press, Cambridge, MA, 2000.

[17] Rocco A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.