An Integer Optimization Approach to Associative Classification

Dimitris Bertsimas **Allison Chang** **Cynthia Rudin**
Operations Research Center
Massachusetts Institute of Technology
Cambridge, MA 02139
dbertsim@mit.edu aachang@mit.edu rudin@mit.edu

Abstract

We aim to design classifiers that have the interpretability of association rules yet have predictive power on par with the top machine learning algorithms for classification. We propose a novel mixed integer optimization (MIO) approach called Ordered Rules for Classification (ORC) for this task. Our method has two parts. The first part mines a particular frontier of solutions in the space of rules, and we show that this frontier contains the best rules according to a variety of interestingness measures. The second part learns an optimal ranking for the rules to build a decision list classifier that is simple and insightful. We report empirical evidence using several different datasets to demonstrate the performance of this method.[1] [2]

1 Introduction

Our goal in this work is to develop classification models that are on par in terms of accuracy with the top classification algorithms, yet are interpretable, or easily understood, by humans. This work thus addresses a dichotomy in the current state-of-the-art for classification: On the one hand, algorithms such as support vector machines (SVM) [1] are highly accurate but not interpretable; for instance, trying to explain a support vector kernel to a medical doctor is not likely to persuade him or her to use an SVM-based diagnostic system. On the other hand, algorithms such as decision trees [2, 3] are interpretable, but not specifically optimized to achieve the highest in-sample accuracy. Our models are both interpretable and directly optimized for accuracy, and can be used for applications in which the user needs accurate predictions as well as an understanding of how the predictions are made.

Our models are designed to be interpretable from multiple perspectives. First, the models are designed to be *convincing*: for each prediction, the model also provides the reasons for why this particular prediction was made, highlighting exactly which data were used to make that prediction. To achieve this, we use association rules to build the models into a type of decision list, that is, a rank ordered set of rules supported by data. The second way our models are interpretable involves their size: these models are designed to be *concise*. Specifically, our formulations include two types of regularization. The first encourages rules to have small left-hand-sides, so that the reasons given for each prediction are as sparse as possible. The second encourages the decision list to be shorter; the regularization term is the number of rules in the decision list, which is another form of sparsity regularization. There is no single correct way to measure interpretability, as it is necessarily subjective. Nevertheless, psychologists have long studied human ability to process data, and have shown that humans can simultaneously process only a handful of cognitive entities, and are able to estimate relatedness of only a few variables [e.g., 4, 5]. We aim in this work to achieve a convincing and concise model that captures relationships between variables, which limits the reasoning

[1] The authorship sequence is alphabetical.

[2] This work was supported by NSF Grant IIS-1053407.

required by humans to understand and believe its predictions. These models allow predictions to be communicated in words, rather than in equations.

The accuracy of our algorithm results from the use of mixed integer optimization (MIO). Rule learning problems suffer from combinatorial explosion, in terms of both searching through a database for rules and managing a massive pile of potentially interesting rules. A dataset with even a modest number of items can contain thousands of rules, thus making it difficult to find useful ones. Moreover, for a set of L rules, there are $L!$ ways to order them into a decision list. On the other hand, MIO solvers are designed *precisely* to handle combinatorial problems. There has been tremendous progress in MIO hardware and software over the last two decades, and we can now solve large-scale MIO formulations that were impossible only a few years ago. On the other hand, designing an MIO problem is more challenging than designing a linear optimization problem. Our ability to solve an MIO problem depends critically on the strength of the formulation, which is related to the geometry of the set of feasible solutions. In this work, we create MIO formulations for both the problem of mining rules and the problem of learning to rank them, and our experiments show predictive accuracy on a collection of datasets at approximately the same level as some of the top current algorithms in machine learning, including support vector machines with Gaussian kernels, C4.5, and boosted decision trees. This shows that one does not necessarily need to sacrifice accuracy to obtain interpretability, as long as one is willing to take more time to generate a better solution.

In Section 2, we discuss related work. In Section 3, we state our notation and derive an MIO formulation for association rule mining. In Section 4, we present an MIO learning algorithm that uses rules to build a classifier. Sections 5 and 6 demonstrate the accuracy and interpretability respectively of our classifiers. We conclude in Section 7. Note that this paper highlights our key ideas and results, but we include additional information in a longer version of the paper [6], including: an extended related work section, a more general MIO formulation for rule generation, supplementary details about our experiments, and additional examples of interpretability.

2 Related Work

Association rule mining was first introduced by Agrawal et al. [7] for market-basket analysis, where the goal was to discover sets of items that were often purchased together. Since the introduction of the Apriori method [8], various algorithms for rule mining have applied heuristic techniques to traverse the search space of possible rules [9]. Though association rules were originally designed for data exploration, *associative classification* later developed as a framework to use the rules for classification, with algorithms such as CBA, CMAR, and CPAR [10, 11, 12], just to name a few. Methods to build a classifier using a sorted set of association rules fall into two categories: those that predict based on multiple rules, and those that predict based on the highest applicable rule in a ranked list of rules. The first category uses more information by classifying based on a sort of majority vote of applicable rules, but in general has two major disadvantages: first, it ignores the dependency between rules, so two rules that are almost exactly the same have two separate votes instead of one; and second, the model loses interpretability by combining rules together. Models that combine the votes of various rules are similar to the Logical Analysis of Data (LAD) model [13]. The second category of sorted-rule-based classification algorithms produces decision lists [14]. These classifiers are simple to understand and use the highest ranked rules for prediction. However, if the list is not properly ordered, it may not yield an accurate classifier. Decision lists can be created by ordering rules according to an interestingness measure. Alternatively, the ordering of rules can be learned from data, which is the approach we take here. Further related work is presented in [6].

3 Mining Optimal Association Rules

In this section, we describe an MIO method to generate rules for the purpose of binary classification. (The method can be trivially extended to multi-class classification.) We use the following standard notation: let $\mathcal{I} = \{1, \ldots, d\}$ be a set of items, and $X \subseteq \mathcal{I}$ be an itemset. Let $\mathcal{D}$ be a database of itemsets. Each itemset or row in the database is called a transaction, and each transaction has a class attribute in $\{-1, 1\}$. For example, the transactions might be medical patients, the items might be various possible symptoms, and the two classes might be "disease 1" and "disease 2." We want

Table 1: The body X of the rule is in transaction i since (1) and (2) are satisfied.

	j				
	1	2	3	4	5
t_i (1 if item j in transaction i)	1	0	1	1	0
b (1 if item j in body of rule)	1	0	0	1	0

Table 2: Interestingness measures.

Measure	**Definition**	**Measure**	**Definition**
Confidence/Precision	$\frac{s}{s_X}$	Conviction	$\frac{1-s_Y}{1-s/s_X}$
Recall	$\frac{s}{s_Y}$	Laplace Correction	$\frac{ns+1}{ns_X+k}$, k is number of classes
Accuracy	$1 - s_X - s_Y + 2s$	Piatetsky-Shapiro	$s - s_X s_Y$
Lift/Interest	$\frac{s}{s_X s_Y}$		

to find association rules of the form $X \Rightarrow -1$ or $X \Rightarrow 1$; the first rule means that a transaction containing X is in class -1, and the second means that a transaction containing X is in class 1.

Let there be n transactions in the database $\mathcal{D}$, and let $t_i \in \{0,1\}^d$ represent transaction i. In particular, $t_{ij} = \mathbf{1}_{[\text{transaction } i \text{ includes item } j]}$ for $1 \le i \le n$ and $1 \le j \le d$. Note that the t_i are data rather than decision variables in the optimization problem.

There are two sets of decision variables. First, let $b \in \{0,1\}^d$ represent the body X of a rule: $b_j = \mathbf{1}_{[j \in X]}$ for $j = 1, \ldots, d$. Second, let $x_i = \mathbf{1}_{[\text{transaction } i \text{ includes } X]}$ for $i = 1, \ldots, n$. Let e_d be the d-vector of ones. Each point in the space $\mathcal{P}$ defined by the following constraints corresponds to the body X of a rule (take $X = \{j : b_j = 1\}$ to obtain X from a feasible b):

$$x_i \le 1 + (t_{ij} - 1)b_j, \quad \forall i, j, \tag{1}$$

$$x_i \ge 1 + (t_i - e_d)^T b, \quad \forall i, \tag{2}$$

$$b_j \in \{0,1\}, \quad \forall j, \tag{3}$$

$$0 \le x_i \le 1, \quad \forall i. \tag{4}$$

To understand (1), consider the two cases $b_j = 0$ and $b_j = 1$. If $b_j = 0$, then the constraint is just $x_i \le 1$, so the constraint has no effect. If $b_j = 1$, then the constraint is $x_i \le t_{ij}$. That is, if $b_j = 1$ (item j is in X) but $t_{ij} = 0$ (item j is not in transaction i), then $x_i = 0$. This set of constraints implies that $x_i = 0$ if transaction i does not include X. We need (2) to enforce $x_i = 1$ if transaction i includes X. Note that $t_i^T b$ is the number of items in the intersection of transaction i and X, and $e_d^T b$ is the number of items in X. This constraint is valid because $t_i^T b = \sum_{j=1}^d t_{ij} b_j \le \sum_{j=1}^d b_j = e_d^T b$, where equality holds if and only if transaction i includes X and otherwise $t_i^T b \le e_d^T b - 1$. Table 1 helps to clarify (1) and (2).

The space $\mathcal{P}$ defined by (1) through (4) has d binary variables, n continuous variables, and $nd + n$ constraints. Here we explain why we do not need an explicit integrality constraint on the x_i variables, that is, why we have (4) instead of $x_i \in \{0,1\}$ for all i. There are two cases when deciding whether X is in transaction i. If it is, then (2) says $x_i \ge 1$, which implies $x_i = 1$. If it is not, then there exists j such that $t_{ij} = 0$ and $b_j = 1$, so (1) says $x_i \le 0$ for some j, which implies $x_i = 0$. Thus in either case, x_i is forced to be an integer, regardless of whether we specify it as an integer variable. Having fewer integer variables generally helps speed up computation.

Our algorithm outputs one rule at a time, for a specified class attribute. Let $y \in \{-1, 1\}$ be the class for which we are mining rules, and let $S = \{i : \text{ transaction } i \text{ has class label } y\}$. Also, let

$$s_X = \frac{1}{n}\sum_{i=1}^n x_i, \quad s_Y = \frac{1}{n}|S|, \quad s = \frac{1}{n}\sum_{i \in S} x_i,$$

called *coverage*, *prevalence*, and *support* respectively. Note that all rules for a given class have the same s_Y. We can capture other interestingness measures using s_X, s_Y, and s, some of which are listed in Table 2.

Many interestingness measures, including those in Table 2, increase with decreasing s_X (holding s constant) and increasing s (holding s_X constant). Thus the rules that optimize each of these

measures fall along an efficient frontier of rules with maximal s and minimal s_X. We can find each rule on the frontier by putting an upper bound on s_X and maximizing s. Formulation (5) maximizes the "scaled support" ($n \cdot s$) for a certain choice of $\bar{s}_X$, where $\bar{s}_X$ denotes the user-specified upper bound on the "scaled coverage" ($n \cdot s_X$). We vary the upper bound over all possible values from largest to smallest to produce the entire frontier (from right to left).

$$\begin{aligned} \max_{b,x} \quad & \sum_{i \in S} x_i - R_{\text{gen\_x}} \sum_{i=1}^{n} x_i - R_{\text{gen\_b}} \sum_{j=1}^{d} b_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq \bar{s}_X, \\ & (b, x) \in \mathcal{P}. \qquad \text{(defined in (1), (2), (3), (4))} \end{aligned} \tag{5}$$

The first term in the objective is the scaled support. The second term corresponds to the coverage s_X; if there are multiple rules with optimal support, we want those with smaller coverage. The third term is a regularization term, and corresponds to the sparsity of the rule; if there are multiple rules that maximize s and have equal s_X, we want those with smaller bodies, that is, more zeros in b. The parameters $R_{\text{gen\_x}}$ and $R_{\text{gen\_b}}$ control the weight of these terms in the objective, where the former ensures that we properly trace out the frontier, and the latter could potentially trade-off sparsity for closeness to the frontier.

Solving (5) once for each possible value of $\bar{s}_X$ does not yield the entire frontier since there may be multiple optimal rules at each point on the frontier. To find other optima, we add constraints making each solution found so far infeasible, so that they cannot be found again when we re-solve. Specifically, we iteratively solve the formulation as follows: Let b^* be the first optimum we find for (5). We add the constraint

$$\sum_{j:b_j^*=0} b_j + \sum_{j:b_j^*=1} (1 - b_j) \geq 1 \tag{6}$$

to the formulation. This constraint says that in the vector b, at least one of the components must be different from in the previous solution; that is, at least one of the zeros must be a one or one of the ones must be a zero. Then we solve again. If we find another optimum, then we repeat the step above to generate another constraint and re-solve. If the optimal value of $\sum_{i \in S} x_i$ decreases, then we set the upper bound on $\bar{s}_X$ to a new value and iterate again. This new value is the minimum of $\sum_{i=1}^{n} x_i$ and $\bar{s}_X - 1$ (previous bound minus one); we know that no rule on the remainder of the frontier has scaled coverage greater than $\sum_{i=1}^{n} x_i$, so using this as the bound provides a tighter constraint than using $\bar{s}_X - 1$ whenever $\sum_{i=1}^{n} x_i < \bar{s}_X - 1$. Using a similar method, we could also find a band of rules below the frontier if we wanted to expand our set of rules.

The rule generation algorithm, called "RuleGen" is summarized in Figure 1. This algorithm allows optional minimum coverage thresholds $\texttt{mincov}_{-1}$ and $\texttt{mincov}_1$ to be imposed on each of the classes of rules. Also, `iter_lim` limits the number of times we iterate the procedure above with adding (6) between iterates for a fixed value of s_X. To find all rules on the frontiers, set $\texttt{mincov}_{-1} = \texttt{mincov}_1 = 0$ and $\texttt{iter\_lim} = \infty$. Figure 2 illustrates the steps of the algorithm.

In [6], we present a formulation for mining general association rules of the form $X \Rightarrow Y$, where Y can be any itemset that is disjoint with X, instead of a class attribute.

4 Building a Classifier

Suppose we have generated L rules, where each rule ℓ is of the form $X_\ell \Rightarrow -1$ or $X_\ell \Rightarrow 1$. Our task is now to rank them into a decision list for classification. Again for ease of exposition, we consider binary classification, though the method extends to multi-class problems. Given a new transaction, the decision list classifies it according to the highest ranked rule ℓ such that X_ℓ is in the transaction, or the highest rule that "applies" to the transaction. In this section, we derive an empirical risk minimization algorithm using MIO that yields an optimal ranking of rules. That is, the ranking returned by our algorithm optimizes the (regularized) classification accuracy on a training sample.

We always include in the set of rules to be ranked two "null rules:" $\emptyset \Rightarrow -1$, which predicts class -1 for any transaction, and $\emptyset \Rightarrow 1$, which predicts class 1 for any transaction. In the final ranking, the

Input: $\texttt{mincov}_{-1}$, $\texttt{mincov}_1$, $\texttt{iter\_lim}$
for Y **in** {-1,1} **do**
  Initialize $\bar{s}_X \leftarrow n$, iter $\leftarrow 1$, $\bar{s} \leftarrow 0$
  Initialize collection of rule bodies $\mathcal{R}_Y = \emptyset$
  repeat
    if iter = 1 **then**
      Solve (5) to obtain rule $X \Rightarrow Y$
      $\bar{s} \leftarrow \sum_{i \in S} x_i$
      iter $\leftarrow$ iter + 1
    end if
    $\mathcal{R}_Y \leftarrow \mathcal{R}_Y \cup X$
    Add new constraint (6)
    if iter $\leq$ $\texttt{iter\_lim}$ **then**
      Solve (5) to obtain rule $X \Rightarrow Y$
      if $\sum_{i \in S} x_i < \bar{s}$ **then**
        $\bar{s}_X \leftarrow \min\left(\sum_{i=1}^{n} x_i, \bar{s}_X - 1\right)$
        iter $\leftarrow 1$
      else
        iter $\leftarrow$ iter + 1
      end if
    else
      $\bar{s}_X \leftarrow \bar{s}_X - 1$
      iter $\leftarrow 1$
    end if
  until $\bar{s}_X < n \cdot \texttt{mincov}_Y$
end for

Figure 1: RuleGen Algorithm.

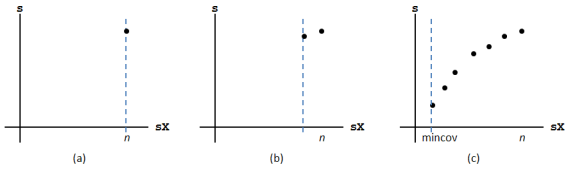

Figure 2: Decrease upper bound (dashed vertical line) starting from $\bar{s}_X = n$ to generate the frontier, one point at a time, from right to left.

Parameters:

$$p_{i\ell} = \begin{cases} 1 & \text{if rule } \ell \text{ correctly classifies transaction } i, \\ -1 & \text{if rule } \ell \text{ incorrectly classifies transaction } i, \\ 0 & \text{if rule } \ell \text{ does not apply to transaction } i, \end{cases}$$

$v_{i\ell} = \mathbf{1}_{[\text{rule } \ell \text{ applies to transaction } i]} = |p_{i\ell}|$,
$R_{\text{rank}} =$ regularization parameter

Variables:
$r_\ell =$ rank of rule ℓ,
$r_* =$ rank of higher null rule,
$u_{i\ell} = \mathbf{1}_{[\text{rule } \ell \text{ is the rule that predicts the class of transaction } i]}$

Figure 3: Parameters and decision variables.

higher of the null rules corresponds effectively to the bottom of the ranked list of rules; all examples that reach this rule are classified by it, thus the class it predicts is the default class. We include both null rules in the set of rules because we do not know which of them would serve as the better default, that is, which would help the decision list to achieve the highest possible classification accuracy; our algorithm learns which null rule to rank higher.

Figure 3 shows the parameters and decision variables of the formulation we derive here to rank a list of rules. The r_ℓ variables store the ranks of the rules; r_* is the rank of the default rule, which we want to be high for conciseness. The $u_{i\ell}$ variables help capture the mechanism of the decision list, enforcing that only the highest applicable rule predicts the class of a transaction: for transaction i, $u_{i\ell} = 0$ for all except one rule, which is the one, among those that apply, with the highest rank r_ℓ. The formulation we designed to build the optimal classifier is:

$$\max_{r, r_*, g, u, s, \alpha, \beta} \quad \sum_{i=1}^{n} \sum_{\ell=1}^{L} p_{i\ell} u_{i\ell} + R_{\text{rank}} r_* \tag{7}$$

$$\text{s.t.} \quad \sum_{\ell=1}^{L} u_{i\ell} = 1, \quad \forall i, \tag{8}$$

$$g_i \geq v_{i\ell} r_\ell, \quad \forall i, \ell, \tag{9}$$

$$g_i \leq v_{i\ell} r_\ell + L(1 - u_{i\ell}), \quad \forall i, \ell, \tag{10}$$

$$u_{i\ell} \geq 1 - g_i + v_{i\ell} r_\ell, \quad \forall i, \ell, \tag{11}$$

$$u_{i\ell} \leq v_{i\ell}, \quad \forall i, \ell, \tag{12}$$

$$\sum_{k=1}^{L} s_{\ell k} = 1, \quad \forall \ell, \tag{13}$$

$$\sum_{\ell=1}^{L} s_{\ell k} = 1, \quad \forall k, \tag{14}$$

$$r_\ell = \sum_{k=1}^{L} k s_{\ell k}, \quad \forall \ell, \tag{15}$$

$$r_* \geq r_A, \tag{16}$$

$$r_* \geq r_B, \tag{17}$$

$$r_* - r_A \leq (L-1)\alpha, \tag{18}$$

$$r_A - r_* \leq (L-1)\alpha, \tag{19}$$

$$r_* - r_B \leq (L-1)\beta, \tag{20}$$

$$r_B - r_* \leq (L-1)\beta, \tag{21}$$

$$\alpha + \beta = 1, \tag{22}$$

$$u_{i\ell} \leq 1 - \frac{r_* - r_\ell}{L-1}, \quad \forall i, \ell, \tag{23}$$

$$\alpha, u_{i\ell}, s_{\ell k} \in \{0, 1\}, \quad \forall i, \ell, k,$$

$$0 \leq \beta \leq 1,$$

$$r_\ell \in \{1, 2, \ldots, L\}, \quad \forall \ell.$$

We use (7) to refer to the entire MIO formulation and not just the objective function. The first term in the objective corresponds to classification accuracy. Given an ordering of rules, the quantity $c_i = \sum_{\ell=1}^{L} p_{i\ell} u_{i\ell}$ equals 1 if the resulting decision list correctly predicts the class of transaction i and -1 otherwise. Thus, the number of correct classifications is $\sum_{i=1}^{n} \left(\frac{c_i+1}{2}\right) = \frac{1}{2}\left(n + \sum_{i=1}^{n} c_i\right)$. So to maximize classification accuracy, it suffices to maximize $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \sum_{\ell=1}^{L} p_{i\ell} u_{i\ell}$. Table 3 shows an example of the parameters ($p_{i\ell}$) and variables (r_ℓ, $u_{i\ell}$) for a particular ranking of rules and transaction to be classified. We note that since this algorithm directly optimizes the 0-1 classification error, it has the property of being robust to outliers.

Table 3: Transaction t_i is represented by $\{1\ 0\ 1\ 1\ 0\}$, and its class is -1. The highest rule that applies is the one ranked 8th ($r_\ell = 8$) since $\{1\ 0\ 1\ 0\ 0\} \subset \{1\ 0\ 1\ 1\ 0\}$ (the rules ranked 10th and 9th do not apply). Thus $u_{i\ell} = 1$ for this rule. This rule has $p_{i\ell} = 1$ since the rule applies to t_i and correctly predicts -1, so the contribution of transaction i to the accuracy part of the objective in (7) is $\sum_{\ell=1}^{L} p_{i\ell} u_{i\ell} = 1$.

Transaction t_i: $\{\mathbf{1\ 0\ 1\ 1\ 0}\}$, **class=**$\mathbf{-1}$			
Ranked rules	$p_{i\ell}$	r_ℓ	$u_{i\ell}$
$\{0\ 1\ 0\ 0\ 1\} \Rightarrow -1$	0	10	0
$\{0\ 1\ 1\ 0\ 0\} \Rightarrow 1$	0	9	0
$\{\mathbf{1\ 0\ 1\ 0\ 0}\} \Rightarrow \mathbf{-1}$	**1**	**8**	**1**
$\{1\ 0\ 0\ 0\ 1\} \Rightarrow -1$	0	7	0
$\{0\ 0\ 0\ 0\ 0\} \Rightarrow 1$	-1	6	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\{0\ 0\ 1\ 1\ 0\} \Rightarrow -1$	1	1	0

Constraint (8) enforces that for each i, only one of the $u_{i\ell}$ variables equals one while the rest are zero. To capture the definition of $u_{i\ell}$, we use an auxiliary variable g_i, which represents the rank of the highest applicable rule for transaction i. Through (9) and (10), there is only one ℓ such that $u_{i\ell} = 1$ is feasible, namely the ℓ corresponding to the highest value of $v_{i\ell} r_\ell$. Constraints (11) and (12) are not necessary but help improve the linear relaxation and thus are intended to speed up computation. We assign the integral ranks r_ℓ using (13) through (15), which imply $s_{\ell k} = 1$ if rule ℓ is assigned to rank k. The matching between ranks and rules is one-to-one.

We add a new type of regularization to favor a shorter overall list of rules by pulling the rank of the higher null rule as high as possible. If r_A is the rank of $\emptyset \Rightarrow -1$ and r_B is the rank of $\emptyset \Rightarrow 1$, then we add r_* to the objective function, where r_* is the maximum of r_A and r_B. The regularization coefficient of r_* in the objective is R_{rank}. We capture r_* using (16) through (22): Either $\alpha = 1$ and $\beta = 0$, or else $\beta = 1$ and $\alpha = 0$. If $\alpha = 1$, then $r_* = r_B$. If $\beta = 1$, then $r_* = r_A$. Since we are maximizing r_*, we know r_* equals the higher of r_A and r_B. Note that if α is binary, then β need not be binary because the constraint $\alpha + \beta = 1$ forces integral values for β. If the rank r_ℓ of rule ℓ is below r_*, then $u_{i\ell} = 0$ for all i, so (23) is valid, and we include it to help speed up computation.

The Ordered Rules for Classification (ORC) algorithm consists of generating rules using RuleGen, computing the $p_{i\ell}$ and $v_{i\ell}$, and then solving the formulation in (7). Note that RuleGen and (7) can be used independently of each other if one desires to use the rules for a different purpose, or to use a set of already established rules to construct the decision list.

5 Computational Results

We used a number of publicly available datasets to demonstrate the performance of our approach. Crime1 and Crime2 are derived from a study funded by the US Department of Justice [15]. Titanic is from a report on the sinking of the "Titanic" [16]. All others are from the UCI Machine Learning Repository [17]. For each dataset, we divided the data evenly into three folds and used each fold in turn as a test set, training each time with the other two folds. The training and test accuracy were averaged over these three folds. We compared the ORC algorithm with six other classification methods—logistic regression, Support Vector Machines (SVM) [1], Classification and Regression Trees (CART) [2], C4.5 [3] (J48 implementation), Random Forests [18], and AdaBoost [19]—all

run using R 2.15.0. We used the radial basis kernel and regularization parameter $C = 1$ for SVM (results for other C values are in [6]), and decision trees as base classifiers for AdaBoost. The ORC algorithm was implemented using ILOG AMPL 11.210 with the Gurobi solver.

Here we explain how we chose the parameter settings for the ORC experiments. In generating rules with (5), we wanted to ensure that $R_{\text{gen\_x}}$ was small enough that the solver would never choose to decrease the scaled support $\sum_{i \in S} x_i$ just to decrease the scaled coverage $\sum_{i=1}^{n} x_i$. That is, $R_{\text{gen\_x}}$ should be such that we would not sacrifice maximizing s for lower s_X; this required only that this parameter be a small positive constant, so we chose $R_{\text{gen\_x}} = \frac{0.1}{n}$. Similarly, we did not want to sacrifice maximizing s or lowering s_X for greater sparsity, so we chose $R_{\text{gen\_b}} = \frac{0.1}{nd}$. In order to not sacrifice classification accuracy for a shorter decision list in ranking the rules with (7), we chose $R_{\text{rank}} = \frac{1}{L}$. We also used $\texttt{mincov}_{-1} = \texttt{mincov}_1 = 0.05$ and $\texttt{iter\_lim} = 5$.

Table 4 shows the average training and test classification accuracy for each dataset; corresponding standard deviations are in [6]. Bold indicates the highest average in the row. Table 5 shows the dataset sizes as well as average number of rules generated by RuleGen and average runtimes for our algorithms ($\pm$one standard deviation); runtimes for the other methods were too small to be a significant factor in assessment. Time_1 is the total time for generating all rules; Time_2 is the time when the final solution for (7) was found, either before solving to optimality or before being terminated after a specified time limit. We generally terminated the solver before (7) solved to provable optimality. Note that often an MIO solver finds an optimum quickly but takes a much longer time to prove optimality, thus terminating early *does not imply* that we do not have an optimum. These results show that in terms of accuracy, the ORC algorithm is on par with top classification methods. The longer version of this paper [6] contains more information on all ORC experiments, including accuracies and runtimes on each fold.

Table 4: Classification accuracy (averaged over three folds).

		LR	SVM	CART	C4.5	RF	ADA	ORC
B.Cancer	train	0.9780	0.9846	0.9561	0.9671	0.9876	0.9693	0.9766
	test	0.9502	**0.9619**	0.9488	0.9590	0.9575	0.9605	0.9532
CarEval	train	0.9580	0.9821	0.9659	0.9907	0.9997	0.9959	0.9598
	test	0.9485	0.9728	0.9618	0.9815	0.9826	**0.9890**	0.9508
Crime1	train	0.8427	0.8439	0.8380	0.8932	0.9918	0.8885	0.8897
	test	0.7394	0.7394	0.7488	0.7465	0.7629	0.7723	**0.7817**
Crime2	train	0.6812	0.7477	0.6858	0.7409	0.8211	0.7156	0.7133
	test	**0.6722**	0.6354	0.6171	0.5941	0.6239	0.6630	0.6699
Haberman	train	0.7712	0.7876	0.7680	0.7745	0.7892	0.7712	0.7680
	test	**0.7582**	0.7386	0.7418	0.7386	0.7386	0.7320	**0.7582**
Mammo	train	0.8482	0.8687	0.8422	0.8596	0.8837	0.8560	0.8536
	test	0.8374	0.8217	0.8301	0.8301	0.8289	**0.8422**	0.8337
MONK2	train	0.6470	0.6736	0.7500	0.9317	0.9907	0.7940	0.8299
	test	0.6019	0.6713	0.6690	**0.8866**	0.6528	0.6389	0.7338
SPECT	train	0.8783	0.8633	0.8390	0.8801	0.9363	0.8839	0.8970
	test	0.7978	**0.8464**	0.7828	0.7940	0.8090	0.8052	0.7753
TicTacToe	train	0.9833	0.9494	0.9348	0.9796	1.0000	0.9917	1.0000
	test	0.9823	0.9165	0.8873	0.9259	0.9781	0.9770	**1.0000**
Titanic	train	0.7783	0.7906	0.7862	0.7906	0.7906	0.7862	0.7906
	test	0.7783	0.7847	0.7846	**0.7906**	0.7833	0.7797	**0.7906**
Votes	train	0.9816	0.9747	0.9598	0.9724	0.9954	0.9701	0.9747
	test	**0.9586**	0.9563	0.9540	**0.9586**	**0.9586**	**0.9586**	0.9563

Table 5: Number of transactions (n), number of items (d), average number of rules generated, average time to generate all rules (Time_1), average time to rank rules (Time_2).

Dataset	n	d	#Rules	Time_1 (sec)	Time_2 (sec)
B.Cancer	683	27	198.3 ± 16.2	616.3 ± 57.8	12959.3 ± 1341.9
CarEval	1728	21	58.0	706.3 ± 177.3	7335.3 ± 2083.7
Crime1	426	41	100.7 ± 15.3	496.0 ± 88.6	12364.0 ± 7100.6
Crime2	436	16	27.3 ± 2.9	59.3 ± 30.4	2546.0 ± 3450.6
Haberman	306	10	15.3 ± 0.6	14.7 ± 4.0	6.3 ± 2.3
Mammo	830	25	58.3 ± 1.2	670.7 ± 34.5	3753.3 ± 3229.5
MONK2	432	17	45.3 ± 4.0	124.0 ± 11.5	5314.3 ± 2873.9
SPECT	267	22	145.3 ± 7.2	71.7 ± 9.1	8862.0 ± 2292.2
TicTacToe	958	27	53.3 ± 3.1	1241.3 ± 38.1	4031.3 ± 3233.0
Titanic	2201	8	24.0 ± 1.0	92.0 ± 15.1	1491.0 ± 1088.0
Votes	435	16	266.0 ± 34.8	108.3 ± 5.0	21505.7 ± 1237.2

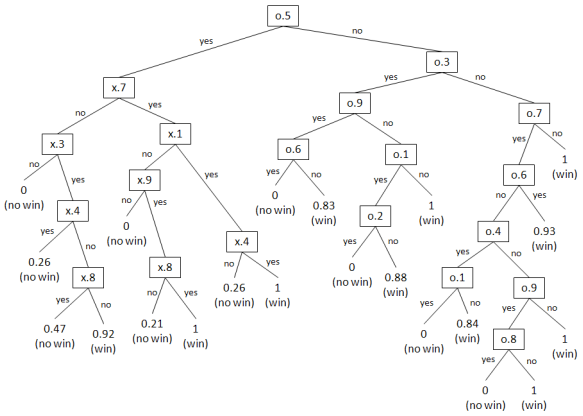

Figure 4: CART classifier for Tic-Tac-Toe data.

1	win			x
	54			x
	54			x

2	win	x		
	61		x	
	61			x

3	win	x	x	x
	42			
	42			

4	win			
	54			
	54	x	x	x

5	win	x		
	57	x		
	57	x		

6	win		x	
	61		x	
	61		x	

7	win			x
	54		x	
	54	x		

8	win			
	55	x	x	x
	55			

9	no win			
	215			
	638			

Figure 5: Ranked rules for Tic-Tac-Toe data (with predicted class, $\bar{s}$, and $\bar{s}_X$ on left side).

6 Interpretability

Interpretability is subjective, but in this section, we aim to demonstrate that the ORC classifier performs well in terms of being easy to understand. Classifiers generated by CART and C4.5 are interpretable because of their decision tree structure. Other methods are not as easily interpreted. For example, the logistic regression model is $p = \frac{1}{1+e^{-\beta_0+\beta^T t}}$, where p is the probability that the class of observation t is 1. The SVM model is a hyperplane that maximizes the margin between the hyperplane and the closest point to it from both classes; by using kernels, we can raise the dimension of the model and achieve high accuracy, but not interpretability. Though there is work devoted to interpreting SVMs, the result is usually a smaller set of nonlinear features, still within a linear combination [20]. AdaBoost combines weak classifiers—decision trees in our experiments—by minimizing an exponential loss function; thus, even though the base classifiers may be interpretable, the final model is not necessarily as interpretable. Random Forests also combines trees.

We give an example using the Tic-Tac-Toe dataset from Section 5. Each point in this dataset represents a board configuration at the end of a Tic-Tac-Toe game where player x played first, and the task is to identify whether player x won. Each of the nine features in the data represents a square on a Tic-Tac-Toe board. The possible values for each feature are: x, o, or blank. Figure 4 shows the CART classifier from training on Folds 1 and 2, which achieves a test accuracy on Fold 3 of 88.1% (average of 88.7% over all folds). The interpretation at the root node is "If there is an 'o' in box 5, then go left, otherwise go right," and similarly for the other nodes. The 21 leaves of the CART tree each predict whether the corresponding path implies a winning board configuration for 'x'. The C4.5 tree achieves a higher test accuracy of 92.8% on Fold 3 (average of 92.6% over all folds), but is even larger with 36 leaves, and is thus less interpretable. The ORC classifier, shown in Figure 5, turns out to just use nine rules. It decides the class of a board the same way a typical human would: if the board has three x's in a line, which can occur in eight different configurations, then player x wins; otherwise, player x does not win. It achieves perfect training and test accuracy.

There are several additional examples of the interpretability of the ORC classifier in [6]. We show that in general, ORC produces a consistently concise model compared with C4.5. The ORC models tend to be larger than the CART trees, but are also more accurate.

7 Conclusion

Our computational experiments show that ORC competes well in terms of training and test accuracy against the top classification algorithms on a variety of datasets. Since our paper is among the first to use MIO methods for machine learning, and in particular to create decision lists using exact approaches, it opens the door for further research on how to use optimization-based approaches for rule mining, forming interpretable classifiers, and handling new forms of regularization. The bottom line is that there is not necessarily a trade-off between accuracy and interpretability. It is truly possible to have both.

References

[1] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, NY, 1995.

[2] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[3] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[4] George A. Miller. The magical number seven, plus or minus two: Some limits to our capacity for processing information. *The Psychological Review*, 63(2):81–97, 1956.

[5] Dennis L. Jennings, Teresa M. Amabile, and Lee Ross. Informal covariation assessments: Data-based versus theory-based judgements. In *Judgment Under Uncertainty: Heuristics and Biases*, pages 211–230. Cambridge Press, Cambridge, MA, 1982.

[6] Dimitris Bertsimas, Allison Chang, and Cynthia Rudin. Ordered rules for classification: A discrete optimization approach to associative classification. *MIT DSpace, Operations Research Center*, 2012. Working paper available at http://dspace.mit.edu/handle/1721.1/73166.

[7] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[8] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499, 1994.

[9] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15:55–86, 2007.

[10] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 80–96, 1998.

[11] Wenmin Li, Jiawei Han, and Jian Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. *IEEE International Conference on Data Mining*, pages 369–376, 2001.

[12] Xiaoxin Yin and Jiawei Han. CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 331–335, 2003.

[13] 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.

[14] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[15] Mark E. Courtney and Gretchen Ruth Cusick. Crime during the transition to adulthood: How youth fare as they leave out-of-home care in Illinois, Iowa, and Wisconsin, 2002-2007. Technical report, Ann Arbor, MI: Inter-university Consortium for Political and Social Research [distributor], 2010.

[16] British Board of Trade. Report on the loss of the 'Titanic' (s.s.). In *British Board of Trade Inquiry Report (reprint)*. Gloucester, UK: Allan Sutton Publishing, 1990.

[17] A. Asuncion and D.J. Newman. UCI Machine Learning Repository, 2007.

[18] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

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

[20] Sören Sonnenburg, Gunnar Rätsch, and Christin Schäfer. Learning interpretable SVMs for biological sequence classification. *Research in Computational Molecular Biology*, 3500(995):389–407, 2005.