Learning customized and optimized lists of rules with mathematical programming

Noname manuscript No.
(will be inserted by the editor)

Learning Customized and Optimized Lists of Rules with Mathematical Programming

Cynthia Rudin · Şeyda Ertekin

Received: date / Accepted: date

Abstract We introduce a mathematical programming approach to building rule lists, which are a type of interpretable, nonlinear, and logical machine learning classifier involving IF-THEN rules. Unlike traditional decision tree algorithms like CART and C5.0, this method does not use greedy splitting and pruning. Instead, it aims to fully optimize a combination of accuracy and sparsity, obeying user-defined constraints. This method is useful for producing non-black-box predictive models, and has the benefit of a clear user-defined tradeoff between training accuracy and sparsity. The flexible framework of mathematical programming allows users to create customized models with a provable guarantee of optimality. The software reviewed as part of this submission was given the DOI (Digital Object Identifier) **10.5281/zenodo.1344142**.

Keywords mixed-integer programming, decision trees, decision lists, sparsity, interpretable modeling, associative classification
68T05 - Computer Science, Artificial Intelligence, Learning and Adaptive Systems

1 Introduction

Our goal is to produce non-black-box machine learning models comprised of logical IF-THEN rule statements. We want these models to be easily customizable by the user, and to have training performance that is optimized. We chose the framework of mathematical programming for this task, which has several benefits over classical techniques for decision trees, decision lists, and other associative classifiers. Users can easily include their own constraints in order to customize the models, and the models are optimal with respect to a given objective. In contrast, classical decision tree and decision list algorithms are generally not guaranteed to produce optimal trees. They use greedy splitting and pruning methods in order to provide models quickly on extremely large datasets. A greedy method that makes the wrong choice at the top of the tree might need to spend the rest of its decisions along the tree compensating for the mistake at the top, without ever being able to correct it. This impacts both accuracy and sparsity of the trees. Because there is no direct control over accuracy or sparsity through splitting and pruning processes, it is difficult to globally optimize accuracy or sparsity using these processes. Decision tree algorithms tend to work poorly on imbalanced problems, leading to a subfield that aims to fix this (e.g. Cieslak and Chawla, 2008). Worse still, there is no clear way to customize a tree to handle other sorts of goals or constraints that a user might have, such as regularizing a non-linear function of model sparsity, or ensuring that the false positive rate is not more than 20%. Without a suitable optimization technique, one may never find a feasible model let alone a useful model. CART (Breiman et al, 1984) seems to be the method of choice for interpretable modeling in industry because of its generally good tradeoff between accuracy and sparsity, despite flawed solutions in many cases. Other decision tree methods like

C. Rudin
Departments of Computer Science, Electrical and Computer Engineering, and Statistical Science, Duke University
E-mail: cynthia@cs.duke.edu

Ş. Ertekin
Department of Computer Engineering, Middle Eastern Technical University, and MIT Sloan School of Management, Massachusetts Institute of Technology
E-mail: seyda@mit.edu

C4.5 (Quinlan, 1993) and C5.0 (Kuhn et al, 2012) tend to produce much larger trees, and lose accuracy when pruned enough to produce an interpretable model. Any method that can guarantee accuracy and sparsity of a logical model (like a tree) could have a direct impact on high stakes decision-making processes where transparency is important. Further, when these models are both interpretable and flexible enough to handle user constraints, this could make the difference between a model that is actually used, and one that sits on a shelf (Ridgeway, 2013). Constructing such models is the problem we consider in this work.

An example rule list is given below. The goal is to predict whether the emergency room will be at capacity or under capacity today.

IF	Saturday=True and summer=True	THEN at capacity(15/20)
ELSE IF	at capacity yesterday=True	THEN at capacity(30/35)
ELSE IF	football on TV today=True	THEN under capacity(7/7)
ELSE IF	weather=ice/snow	THEN at capacity(40/43)
ELSE		under capacity(42/46).

The selection and order of these rules determines accuracy. The fractions reported between parentheses are empirical conditional probabilities of making a correct prediction for a given rule. For instance, the empirical conditional probability 30/35 for the second rule means that 35 observations did not satisfy the first rule, but satisfied the second rule. Of those, in 30 observations the emergency room was at capacity. The order of rules is important because in a given situation, the rule that makes the prediction is the first rule that applies to it.

Our method for producing rule lists works in two steps, like an associative classification technique: first, in Step 1, a set of "IF-THEN" association rules with high support and high confidence are mined using a rule-mining algorithm, and second, in Step 2 we learn the ordering of the rules from data to form a sparse and accurate decision list. We use discrete optimization techniques that are designed to handle the combinatorial explosion caused by a combinatorially increasing number of possible rules, and an exponentially increasing number of ways to order them. We provide two mathematical programming formulations: one for mining the rules, and the other for ordering them. Each of the two formulations can be used independently; for instance it is possible to use our formulation for learning the ordering of rules when the rules themselves were generated from a separate rule mining algorithm. Alternatively, it is possible to use our rule mining algorithm for examining rules that might be individually interesting. Both are mixed-integer *linear* programs: constraints and the objective are linear in the decision variables.

Our method has the same advantages as any other algorithm that subdivides the feature space in that it is robust to outliers in the data, and handles missing data naturally, without imputation. Computation scales with the number of rules mined, *not* the size of the dataset. That is, computation tends to scale inversely with the sparsity of the dataset, not its size.

The organization of this paper is as follows: Section 2 briefly reviews the relationship to classical approaches to constructing decision trees, decision lists, and other associative classifiers. Section 3 provides the main algorithm for ordering rules and an example of the optimal solution. Section 4 includes a proposed rule mining approach. Section 5 discusses possible ways to add cuts to tighten the feasible region for the main rule-ordering algorithm. Section 6 provides experimental results on a variety of datasets. Section 7 discusses a few ways to customize rule lists. Section 8 provides related work and discussion.

This work follows a line of work on optimal rule lists, motivated by Bayesian approaches (Rudin et al, 2011, 2013; Letham et al, 2015; Yang et al, 2017; Wang and Rudin, 2015), with earlier closely related ideas in the PhD thesis of Allison Chang (Chang, 2012), with direct follow-up work to this paper in Angelino et al (2017, 2018) where some of this paper's bounds are used for a branch-and-bound algorithm, with follow-up to that work in Chen and Rudin (2018).

2 Relationship to Classical Methods

The classical way to construct a decision tree is using greedy splitting followed by greedy pruning. For instance, let us continue the example above, where we predict whether the emergency room will be at capacity. A typical decision tree method like CART (Breiman et al, 1984) or C4.5 (Quinlan, 1993) would consider the full dataset, and select a feature to "split" on to form the top of the tree. The splitting criteria is either typically either information gain or Gini index. For instance, we might split on the feature "at capacity yesterday" if we find that the information gain for splitting on that feature is the

largest out of all features. In that case, the data would be split into two pieces, based on whether the hospital capacity was reached yesterday, and each piece would be handled separately. Within each of the two pieces, the algorithm would again test for which feature to split on, and continue recursively splitting until the node is pure, meaning that no more splits are possible since all of the labels within the leaf are identical. At this point, we have a tree constructed in a fully greedy way, with local optimization of information gain (or Gini index). The tree fits the training data perfectly, since all of the nodes are pure. To prevent overfitting, the algorithm then has a greedy pruning stage, where it aims to greedily remove nodes that do not classify enough points accurately. This yields the final tree.

A typical decision list algorithm (Rivest, 1987), associative classification method (Liu et al, 1998), or rule induction method (Cohen, 1995) would perform similar greedy construction; the difference is that in associative classification methods, usually rules are pre-mined, and instead of allowing arbitrary splits, these methods are only allowed to use the pre-mined rules to construct the rule list.

In these approaches, there is no obvious mechanism to introduce user-defined constraints. This means one cannot (in any realistically practical way) use classical decision tree techniques to build a customized model. The parameters of a classical decision tree algorithm alter only the splitting criteria and the pruning criteria. There is no direct mechanism to change the global objective directly, nor is there a direct way to incorporate constraints.

Further, it is not clear how far from optimal a given decision tree is. This makes it very difficult to judge whether a given decision tree from a classical method is the best option, since it is possible that a much better tree exists. Even worse, using classical methods causes confusion over the source of poor performance. If a decision tree from a classical method performs poorly, it is not clear whether the performance is caused by limiting ourselves to using decision trees rather than considering a more flexible form of model class, or whether the poor performance is due to the algorithm's choice of a suboptimal tree. This question arises in cases where there are consequences for using black box models, when one would not want to resort to a black box unless there was a substantial gain in performance for doing so. Work of Angelino et al (2018); Zeng et al (2017), extends the present work to address criminological scoring systems, as there is an industry in the United States for proprietary criminal recidivism prediction scores. Their work show that there is very little prediction accuracy difference between black box machine learning models, the proprietary risk prediction score "COMPAS," and much simpler transparent models.

These issues illustrate where mathematical programming has a clear advantage; it allows us to fully optimize the selection and order of rules to be accurate, sparse, and obey user-defined constraints. There is a tradeoff: classical methods are very fast but not very accurate, and work on large datasets, but it can be difficult to customize them to obey user-defined constraints. The problem of finding an optimal decision tree is NP-hard (Naumov, 1991), so classical methods do not aim to fully solve the problem to optimality. The method in this work is easy to customize (by adding constraints to the MIP), and yields accurate models, but takes longer to run and cannot handle huge datasets. The proposed algorithm has been successfully applied to healthcare and criminology problems. These are problems that do not have huge datasets, require constraints on the model, and for which expert time is more expensive than computational time.

3 Optimized Rule Lists

Formally, a rule list is denoted $f = (L, \{(a_l, \gamma_l)\}_{l=1}^{L})$, where $L \in \mathbb{Z}^+$ is the number of rules in the list, $a_l(\cdot) \in B_x(\cdot)$ for $l = 1 \dots L-1$, where $B_x(\cdot)$ denotes the space of boolean functions on feature space X, and the default ELSE rule is $a_L(x) = \text{True}\ \forall x$. Also γ_l is the predicted outcome (either 0 or 1) for rule l. Notation $f(x)$ means the prediction made by the first rule that x obeys (the rule that captures x), in particular, we have $f(x) = \gamma_l$, where $l = \min l^{'}$ s.t. $a_{l'}(x) = \text{True}$. Thus the rule list f looks like: IF $a_1(x)$ THEN predict γ_1, ELSE IF $a_2(x)$ THEN predict γ_2, ELSE IF $a_3(x)$ THEN predict γ_3,..., ELSE (since a_L is true) predict γ_L.

In this section, we show how a set of rules can be ordered optimally to form a sparse and accurate rule list. This is Step 2 of the procedure for producing rule lists. The rules can come from the rule-mining algorithm in the following section, or they can come from any other rule mining algorithm, to handle Step 1. As long as the rule mining method produces a sufficiently expressive set of rules, the rule-ordering step will be the key to constructing the predictive model.

The input to the rule-ordering algorithm includes a training set of labeled observations $\{(x_i, y_i)\}_{i=1}^{m}$, with observations $x_i \in X$, and labels $y_i \in \{-1, 1\}$ (in the example, "at capacity" might be assigned the label $+1$). The method here can be generalized to the multi-class setting easily. Also given as input is a

collection of rules, which comes from a rule mining algorithm, or if the problem is small enough, we can input all possible rules below a certain size. R is the total number of rules available. Starting from the top of the list, each rule r evaluates each observation x_i to determine whether it applies, meaning that the "IF" condition is met, and if so, the rule assigns a predicted label of either -1 or 1. We define an $m \times R$ matrix of parameters $\mathbf{A}$, called the *applies* matrix:

$$A_{ir} := \begin{cases} 1 & \text{when rule } r \text{ applies to observation } i, \\ 0 & \text{otherwise.} \end{cases}$$

Also we define the $m \times R$ *deciding rule* matrix $\mathbf{d}$, where d_{ir} is a decision variable that is 1 if the rule r is the rule that makes the prediction for observation i, and 0 otherwise. In other words, d_{ir} indicates the first rule in the decision list that applies to observation i. Note that d_{ir} is a function of the permutation of rules. We say that d_{ir} shows which rule r *captures* observation i.

The *permutation of rules* is a decision variable called $\boldsymbol{\pi}$, which is a vector of length R containing each of the values $\{1, ..., R\}$. The entry π_r is the "position" or "height" of rule r in the list. The rule with the highest position R is at the top of the decision list. Matrix $\boldsymbol{\delta}$ is an indicator matrix with entries $\{\delta_{rk}\}_{rk}$ encoding which rule is assigned to which position in the permutation $\boldsymbol{\pi}$, so it encodes the same information as in $\boldsymbol{\pi}$. Variable h_i is the height of the rule that makes the prediction for observation i.

The list of rules must contain the default rules, whose IF condition always holds (e.g., "ELSE under capacity"). Every decision list must end with a default rule, otherwise there could be a portion of the input space that does not have predictions made for it. If a decision list has more than one default rule, the list effectively ends with the first default rule: any observation arriving at that rule gets a prediction. Our algorithm creates a full ordering of the rules, where the portion of the list until the first default rule is sufficient to fully define the predictive model. In our notation, if the rule "ELSE predict +1" is given index $r_+ \in \{1, ..., R\}$ and rule "ELSE predict -1" has index r_-, then the height of the higher of those two rules is denoted $\max_{\tilde{r} \in \{r_+, r_-\}} \pi_{\tilde{r}}$, also encoded by $\tilde{\pi}$ in our formulation. Since r_+ and r_- are within $\{1, ..., R\}$, when the formulation indicates that conditions hold $\forall r$, this includes the default rules r_+ and r_-.

Let us define parameter matrix $\mathbf{M} : \mathbb{R}^m \times \mathbb{R}^R$ element-wise so that: $M_{ir} = 1$ if rule r applies to i and its right hand side agrees with y_i; $M_{ir} = -1$ rule r applies to i and its right hand side disagrees with y_i; and $M_{ir} = 0$ otherwise, when rule r does not apply to observation i.

We define S_r to be the *size* of rule r which is the number of (precomputed) conditions (predicates) in rule r (e.g., "Saturday=True and summer=True" has two conditions). We define β_r to be a binary decision variable indicating whether rule r is in the decision list. This means that $\sum_r S_r \beta_r$ is the number of total conditions in the decision list.

We will choose the permutation of rules to maximize classification accuracy, regularized by the sparsity of the decision list, and regularized by the number of conditions in the rules. Here is the formulation:

$$\max_{\boldsymbol{\pi},\tilde{\pi},\mathbf{h},\mathbf{d},\boldsymbol{\beta},\boldsymbol{\delta},\gamma_+,\gamma_-} \quad \left(\frac{1}{2}m + \frac{1}{2}\sum_{i=1}^{m}\sum_{r=1}^{R} M_{ir}d_{ir}\right) + C\tilde{\pi} - C_1\sum_{r=1}^{R}\beta_r S_r \quad \text{s.t.}$$

$$\begin{aligned}
h_i &\geq A_{ir}\pi_r, \forall i, r, && (1)\\
h_i &\leq A_{ir}\pi_r + R\cdot(1-d_{ir}), \forall i, r, && (2)\\
\sum_{r=1}^{R} d_{ir} &= 1, \forall i, && (3)\\
d_{ir} &\geq 1 - h_i + A_{ir}\pi_r, \forall i, r, && (4)\\
d_{ir} &\leq A_{ir}, \forall i, r, && (5)\\
\pi_r &= \sum_{j=1}^{R} j\delta_{rj}, \forall r, && (6)\\
\sum_{j=1}^{R}\delta_{rj} = 1, \forall r, \text{ also } &\sum_{r=1}^{R}\delta_{rj} = 1, \forall j, && (7)\\
\tilde{\pi} - \pi_{r_+} &\leq (R-1)\gamma_+, && (8)\\
\pi_{r_+} - \tilde{\pi} &\leq (R-1)\gamma_+, && (9)\\
\tilde{\pi} - \pi_{r_-} &\leq (R-1)\gamma_-, && (10)\\
\pi_{r_-} - \tilde{\pi} &\leq (R-1)\gamma_-, && (11)\\
\gamma_+ + \gamma_- &= 1, && (12)\\
\tilde{\pi} \geq \pi_{r_+}, \quad &\text{and} \quad \tilde{\pi} \geq \pi_{r_-}, && (13)\\
\pi_r - \tilde{\pi} &\leq (R-1)\beta_r \forall r && (14)\\
\gamma_-, d_{ir}, \delta_{rj}, \beta_r &\in \{0,1\}, \forall i, r, j, && (15)\\
\pi_r &\in \{1, 2, \ldots, R\}, \forall r. && (16)\\
0 \leq \gamma_+ &\leq 1. && (17)
\end{aligned}$$

Note that the constraints (except (15) and (16)) and objective are *linear* in the decision variables, which is the challenging part of this construction. Having a mixed-integer linear formulation allows one to find a globally optimal solution.

The first term in the objective is $\left(\frac{1}{2}m + \frac{1}{2}\sum_{i=1}^{m}\sum_{r=1}^{R} M_{ir}d_{ir}\right)$ which is the classification accuracy of the permutation of rules π. To see this, note that $\sum_{i=1}^{m}\sum_{r=1}^{R} M_{ir}d_{ir}$ is the number of correct predictions minus the number of incorrect predictions, which equals twice the number of correct predictions minus m. Rearranging, we find the term in the objective:

$$\sum_{i=1}^{m}\sum_{r=1}^{R} M_{ir}d_{ir} = 2(\#\text{Correct}) - m$$

$$\#\text{Correct} = \frac{1}{2}m + \frac{1}{2}\sum_{i=1}^{m}\sum_{r=1}^{R} M_{ir}d_{ir}.$$

The second term in the objective $C\tilde{\pi}$ contains the height of the highest default rule within the decision list, which we want to be as large as possible so that the list is as small as possible (vertical sparsity). The third term contains the total number of conditions in the decision list, which we want to be as small as possible (horizontal sparsity). The variable β_r is 1 if rule r is in the list, so $\beta_r S_r$ is the size of rule r if it is in the list and 0 otherwise.

Constraints (1), (2), (3) together specify the deciding rule matrix $\mathbf{d}$'s and the heights $\mathbf{h}$. According to these three constraints, there can be one feasible height h_i and deciding rule d_{ir} for each example i, which is the rule r that applies and whose height is the highest. This is because (3) states that only one rule can make a decision for observation i, and for whichever rule that is, (1) and (2) then say that $h_i = A_{ir}\pi_r$. Also (1) says that the height h_i of the rule classifying observation i is at least that of the highest rule it applies to. This means that the rule classifying i is exactly the one with the highest height it applies to.

Constraint (4) is a valid cut; it is not necessary but helps to eliminate portions of the search region defined by the relaxation. It states that the deciding rule variable d_{ir} for an observation can only be an allowable value (1 or 0) if the height of the deciding rule is at least as high as all of the rules that apply to it. Constraint (5) is also a valid cut that is not necessary, stating that a rule cannot decide for an observation if it does not apply. It can be useful to have these constraints to reduce computing times by strengthening the LP-relaxation of the model.

Constraint (6) provides the relationship between the permutation of rules and the indicator matrix $\boldsymbol{\delta}$. Again, $\boldsymbol{\delta}$ assigns rules to positions in the permutation. Once $\boldsymbol{\pi}$ is determined, it completely determines $\boldsymbol{\delta}$ and vice versa, they encode the same information. Constraint (7) defines the $\boldsymbol{\delta}$ matrix properly as an indicator matrix.

To define the height of the higher of the two default rules $\tilde{\pi}$, we first ensure that $\tilde{\pi}$ is either π_{r_+} or π_{r_-} in (8)-(12), where only one of γ_+ and γ_- are 1 by rule (12). Then we force $\tilde{\pi}$ to be the higher of π_{r_+} and π_{r_-} using (13).

Constraint (14) defines the β_r's as being 1 when the rule is in the list. This constraint can be removed if the second regularization term is not being used. The remaining constraints, (15) and (16), define the appropriate variables as binary and integer, along with (17), which paired with the fact that γ_- is binary, ensures that γ_+ is also binary.

The formulation above also applies directly to multi-class classification problems. The only change is to include a default rule for each of the additional classes, and there would be an additional set of constraints.

There is an explicit accuracy/sparsity tradeoff with this method because it has a global objective. Each C additional correct predictions are worth the same as one fewer rule in the list. Similarly, each additional C_1 correct predictions are worth the same as one fewer condition (within any rule) in the decision list. The maximum possible value of $\tilde{\pi}$ is R, so, ignoring the third term for now, this means that if the regularization constant C is less than $1/R$, we would not sacrifice any training accuracy for sparsity in the number of rules. Because we can use these parameters to control sparsity, they also control interpretability and overfitting. We will discuss in Section 5.1 another major advantage of the control of sparsity, which is that it tells us the necessary support of pre-mined rules to obtain a globally optimal solution over all possible rules.

We remark that the formulation implies the following:

Proposition 1 *If $\pi_r < \tilde{\pi}$ then $d_{ir} = 0$.*

The proof is that (1) states that $h_i \geq A_{ir}\pi_{ir}\ \forall i, r$. Considering specifically $r = r_+$ and $r = r_-$, (1) becomes $h_i \geq A_{ir_+}\pi_{ir_+}$, and $h_i \geq A_{ir_-}\pi_{ir_-}$. The rules r_+ and r_- apply to all i because they are default rules, so $A_{ir_+} = A_{ir_-} = 1$. Also $\tilde{\pi}$ is the max of π_{r_+} and π_{r_-} from the $\tilde{\pi}$ equations. Putting this together, we have $h_i \geq \tilde{\pi}\ \forall i$. From (1), (2) and (3), we have $d_{ir} = 1 \iff h_i = A_{ir}\pi_r$, thus, $d_{ir} = 1 \iff h_i = A_{ir}\pi_r$ where $\pi_r \geq \tilde{\pi}$. Thus for rules r where $\pi_r < \tilde{\pi}$, it would be a contradiction to have $d_{ir} = 1$ for any i. Thus, d_{ir} must be 0 for all i, for all r such that $\pi_r < \tilde{\pi}$.

This leads to the following corollary.

Corollary 1 *The following equation is valid for the MIP.*

$$d_{ir} \leq 1 - \frac{\tilde{\pi} - \pi_r}{R - 1}, \forall i, r. \tag{18}$$

Constraint (18) states that if the height of a rule is below that of the default rule, it cannot possibly make any decisions on any observations. It could be added to the formulation optionally to speed up computation by strengthening the LP-relaxation of the model.

Next we walk through the variables for a small problem's optimal solution.

3.1 Example of optimal solution to this formulation

Table 1 shows a small example, for the problem of classifying tic tac toe boards as to whether or not the "X" player won the game. The answer, obviously, is that any tic tac toe board where there are three "X"'s in a row should be classified as positive, and all other boards should be classified as negative. The observations are tic tac toe boards after the game has finished, including X, O and blank spaces. Variable $x1$ indicates whether there was an X within the upper left box of the tic tac toe board. The boxes are labeled 1-9 going across from left to right and from top to bottom. We listed the full set of rules we are

Rule ID	Original Order			Rule ID	Ranked Order			Height
1		→	Win	9	$x7\ x8\ x9$	→	Win	13
2		→	No Win	5	$x2\ x5\ x8$	→	Win	12
3	$x1\ x5\ x9$	→	Win	10	$x1\ x2\ x3$	→	Win	11
4	$x3\ x5\ x7$	→	Win	4	$x3\ x5\ x7$	→	Win	10
5	$x2\ x5\ x8$	→	Win	8	$x4\ x5\ x6$	→	Win	9
6	$x1\ x4\ x7$	→	Win	7	$x3\ x6\ x9$	→	Win	8
7	$x3\ x6\ x9$	→	Win	6	$x1\ x4\ x7$	→	Win	7
8	$x4\ x5\ x6$	→	Win	3	$x1\ x5\ x9$	→	Win	6
9	$x7\ x8\ x9$	→	Win	2		→	No Win	5
10	$x1\ x2\ x3$	→	Win	12	$x1\ x7\ x8$	→	No Win	4
11	$x1\ x2\ x4$	→	No Win	1		→	Win	3
12	$x1\ x7\ x8$	→	No Win	13	$x3\ x4$	→	No Win	2
13	$x3\ x4$	→	No Win	11	$x1\ x2\ x4$	→	No Win	1

Table 1 The left columns show rules for the example problem. The right columns show the optimal ordering of the rules after we solve the optimization problem.

using for the small example in the left part of Table 1. Here r_+ is 1 and r_- is 2. In the right part of the figure, we listed the optimal ordering of the rules that we found from running the optimization procedure on the 640 observations in the UCI tic tac toe dataset (Bache and Lichman, 2013). In particular, the rule with index 9 is ranked first, which says "if there are X's in the lower left, lower middle, and lower right of the tic tac toe board (spots 7, 8, and 9), then vote that the X player wins." This rule has height 13 because there are 13 rules. The list effectively ends at rule 2 which is the default rule for "X does not win," which is in height 5. This means the 4 rules below it are not being used. Table 2 shows other decision variables. The $\boldsymbol{\pi}$ vector $[3, 5, 6, ...]$ indicates, for instance, that the rule with index 1 (the default rule for X wins) is in height 3, which one can see from the right side of the chart, where the third last rule is the default for X wins. Variable $\tilde{\pi}$ is 5 because the list effectively ends at the 5th rule from the bottom, which is the default rule for "X does not win." γ_+ is 1, which means $\gamma_- = 0$ by Equation (12), and then by (10) and (11) we know $\tilde{\pi} = \pi_{r_-} = \pi_2$. Vector $\boldsymbol{\beta}$ is 1 for all heights that are high enough to be in the list, meaning it is one for heights above height 5. As it turns out, the value of β_r is irrelevant for the default rules. This is because β_r is multiplied by S_r, the number of conditions in rule r, which is 0 for the default rules. This means β_r is unconstrained and irrelevant for r_+ and r_-, in our case, β_1 and β_2, which happen to be set to 1. We marked those by stars in Table 2. The $\boldsymbol{\delta}$ matrix is also shown, which encodes the same information as $\boldsymbol{\pi}$. Here $\boldsymbol{\pi}$ is shown to the right of $\boldsymbol{\delta}$.

The objective value is shown below, decomposed into its three terms. The first term is 640, because there are 640 observations, and all of them are classified correctly. The second term is $C\tilde{\pi}$ where C was set to $1/13$ because there are 13 rules, and $\tilde{\pi}$ was determined to be 5. The value $1/13$ was chosen so that no training accuracy would be sacrificed to make the list sparser. The value of C_1 in the third term was also set to $1/R = 1/13$, and the total $\sum_{r=1}^{R} \beta_r S_r$ is 24=8 rules × 3 conditions per rule.

$$\begin{aligned} \text{Objective} &= \left(\frac{1}{2}m + \frac{1}{2}\sum_{i=1}^{m}\sum_{r=1}^{R} M_{ir}d_{ir}\right) && +C\tilde{\pi} && -C_1\sum_{r=1}^{R}\beta_r S_r \\ &= 640 && +0.384615 && -1.84615 \quad = 638.53846. \end{aligned}$$

Table 3 shows three of the observations and their decision variables. The first observation has index 1 and has three X's in a row, in positions 3, 5, and 7, along with various O and empty squares (a tic tac toe board can have empty squares when the game ends). The rule that made the decision for this observation is $x3\ x5\ x7 \rightarrow 1$, which is rule 4, hence the $d[1, :]$ vector having element 4 being 1. The height of rule 4 is 10, which is why h[1] is 10. Similar logic follows for the other two observations provided in Table 3.

4 Optimized Rule Mining

We propose a method for mining rules to use for Step 1 in our method, also by formulating it as a mixed integer linear program. Ideally we would choose all rules that have high support in the data, but if that number is too large, we would like to produce a set of rules that are optimally interesting according to a variety of interestingness measures used in the association rule literature (Geng and Hamilton, 2006;

Variable	Index													
	1	2	3	4	5	6	7	8	9	10	11	12	13	
$\boldsymbol{\pi}$	3	5	6	10	12	7	8	9	13	11	1	4	2	
$\tilde{\pi}$	5													
γ_+	1													
γ_-	0													
β	1*	1*	1	1	1	1	1	1	1	1	0	0	0	π
$\delta[1,:]$	0	0	1	0	0	0	0	0	0	0	0	0	0	3
$\delta[2,:]$	0	0	0	0	1	0	0	0	0	0	0	0	0	5
$\delta[3,:]$	0	0	0	0	0	1	0	0	0	0	0	0	0	6
$\delta[4,:]$	0	0	0	0	0	0	0	0	0	1	0	0	0	10
$\delta[5,:]$	0	0	0	0	0	0	0	0	0	0	0	1	0	12
$\delta[6,:]$	0	0	0	0	0	0	1	0	0	0	0	0	0	7
$\delta[7,:]$	0	0	0	0	0	0	0	1	0	0	0	0	0	8
$\delta[8,:]$	0	0	0	0	0	0	0	0	1	0	0	0	0	9
$\delta[9,:]$	0	0	0	0	0	0	0	0	0	0	0	0	1	13
$\delta[10,:]$	0	0	0	0	0	0	0	0	0	0	1	0	0	11
$\delta[11,:]$	1	0	0	0	0	0	0	0	0	0	0	0	0	1
$\delta[12,:]$	0	0	0	1	0	0	0	0	0	0	0	0	0	4
$\delta[13,:]$	0	1	0	0	0	0	0	0	0	0	0	0	0	2

Table 2 Optimal values for decision variables in the example.

Index and observation		Classifying rule	Corresponding d vector	h
1	– – x – x o x o –	$x3\ x5\ x7 \rightarrow 1$	$d[1,:] = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$	$h[1] = 10$
4	o – x o o – x x x	$x7\ x8\ x9 \rightarrow 1$	$d[4,:] = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$	$h[4] = 13$
120	o x o o x x x o x	$\rightarrow 0$	$d[120,:] = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$	$h[120] = 5$

Table 3 Three of the observations and their decision variables.

McGarry, 2005; Tan and Kumar, 2000; Bayardo and Agrawal, 1999). In particular, we will mine rules along a frontier of possible rules (in the spirit of, for instance, Bayardo and Agrawal, 1999, and Hata et al, 2013), along which several well-known interestingness measures are maximized. Let us now define the partial order that identifies the frontier. We will use notation for general rule mining, and later specialize to binary classification. As this topic is separate from the previous one, we will use fresh notation.

The dataset consists of m observations (transactions), each of which is a subset of items. In this case, items are simply predicates that are true. The full set of items is indexed by $I = 1, ..., N$. For rule $a \rightarrow c$, the left side a is called the antecedent, and is a subset of items (an itemset). The right side c is also an itemset, called the consequent, where $a \cap c = \emptyset$. For the rule $a \rightarrow c$, for convenience define a_i to be 1 if observation i includes antecedent a and 0 otherwise. Similarly, $c_i := 1$ if observation i includes the consequent c, and 0 otherwise. The *support* of an itemset a is the fraction of transactions that contain it. The support of the antecedent is $\text{supp}(a) = \sum_i a_i$. The support of the consequent is $\text{supp}(c) = \sum_i c_i$. The support of the rule is $\text{supp}(a \rightarrow c) = \sum_i (a_i c_i)$, that is, the number of transactions for which a_i and c_i are both 1. Define the partial order: $a^{(1)} \rightarrow c^{(1)} \preceq a^{(2)} \rightarrow c^{(2)}$ whenever

$$\text{supp}(a^{(1)} \rightarrow c^{(1)}) \leq \text{supp}(a^{(2)} \rightarrow c^{(2)}), \text{ and}$$
$$\text{supp}(a^{(1)}) \geq \text{supp}(a^{(2)}), \quad \text{and} \quad \text{supp}(c^{(1)}) \geq \text{supp}(c^{(2)}).$$

If any of the inequalities are strict, then we write $a^{(1)} \rightarrow c^{(1)} \prec a^{(2)} \rightarrow c^{(2)}$.

The frontier we will use in this paper is the set of rules that are not dominated by other rules according to the partial order $\prec$. That is, the frontier includes $a^* \rightarrow c^*$ such that there is no rule $a \rightarrow c$ for which $a^* \rightarrow c^* \prec a \rightarrow c$. This frontier is three dimensional; one dimension for the support of a, one for the support of c, and the last for the support of the rule $a \rightarrow c$. Among rules with equal antecedent and consequent support, the partial order prefers rules with higher rule support. This means we prefer rules where the antecedent and consequent are more highly correlated. Many popular interestingness measures

increase with the support of the rule, and decrease with the supports of the antecedent and consequent:

$$\text{Confidence}(a \to c) = \hat{P}(c|a) = \frac{\text{supp}(a \cup c)}{\text{supp}(a)}$$

$$\text{Recall}(a \to c) = \hat{P}(a|c) = \frac{\text{supp}(a \cup c)}{\text{supp}(c)}$$

$$\text{Lift}(a \to c) = \frac{\hat{P}(a \cup c)}{\hat{P}(a)\hat{P}(c)} = \frac{\text{supp}(a \cup c)}{\text{supp}(a)\ \text{supp}(c)}$$

$$\text{Laplace}(a \to c) = \frac{m\hat{P}(a \cup c) + 1}{m\hat{P}(a) + K} = \frac{m\ \text{supp}(a \cup c) + 1}{m\ \text{supp}(a) + K}$$

$$\text{Conviction}(a \to c) = \frac{1 - \hat{P}(c)}{1 - \hat{P}(c|a)} = \frac{1 - \text{supp}(c)}{1 - \frac{\text{supp}(a \cup c)}{\text{supp}(a)}}$$

$$\begin{aligned}\text{Accuracy}(a \to c) &= \hat{P}(a \cup c) + \hat{P}(I \backslash a \cup I \backslash c) \\ &= 1 - \text{supp}(a) - \text{supp}(c) + 2\text{supp}(a \cup c).\end{aligned}$$

$$\begin{aligned}\text{Piatetsky-Shapiro}(a \to c) &= \hat{P}(a \cup c) - \hat{P}(a)\ \hat{P}(c) \\ &= \text{supp}(a \cup c) - \text{supp}(a)\text{supp}(c).\end{aligned}$$

(Note that $\text{supp}(a \cup c)$ means the count of points that obey the union of conditions in a and c, meaning the set of *both* conditions hold; the intersection of conditions a and c is empty.)

If we mine all rules on the frontier, we will have mined all rules where any of these measures are optimized.

4.1 Valid Rules

We need to construct variables and constraints that define what it means for itemsets to form a valid rule. Every rule we mine will obey these constraints.

We index the observations by i, and the items by j. The dataset is represented by a matrix $\mathbf{U}$, where U_{ij} is 1 when transaction i contains item j and 0 otherwise. We would like to uncover a new valid rule $a \to c$ where a and c are non-intersecting itemsets. The antecedent a is represented as a binary vector $\boldsymbol{\alpha}$ of size N, where $\alpha_j = 1$ if a contains item j and 0 otherwise. The consequent c is similarly represented by $\boldsymbol{\gamma} \in \{0,1\}^N$. Notation $\mathbf{1}$ indicates a vector of 1's of length N. In addition to the definitions a_i and c_i above, we define b_i to be 1 if observation i contains *both* the antecedent and consequent, and 0 otherwise. Thus we have created the following definition.

Definition. Any *valid* rule $a \to c$ must obey the following list of constraints:

$$\alpha_j + \gamma_j \leq 1\ \forall j, \text{ and } \alpha_j, \gamma_j \in \{0,1\}\ \forall j, \tag{19}$$

$$a_i \leq 1 + (U_{ij} - 1)\alpha_j, \quad \forall i, j, \tag{20}$$

$$a_i \geq 1 + (\mathbf{U}_{i\cdot} - \mathbf{1})^T \boldsymbol{\alpha}, \quad \forall i. \tag{21}$$

$$c_i \leq 1 + (U_{ij} - 1)\gamma_j, \quad \forall i, j, \tag{22}$$

$$c_i \geq 1 + (\mathbf{U}_{i\cdot} - \mathbf{1})^T \boldsymbol{\gamma}, \quad \forall i, \tag{23}$$

$$b_i \leq a_i, \quad \forall i, \quad \text{and} \quad b_i \leq c_i, \quad \forall i, \tag{24}$$

$$b_i \geq a_i + c_i - 1, \quad \forall i, \tag{25}$$

$$0 \leq a_i, c_i, b_i \leq 1, \quad \forall i. \tag{26}$$

Again, the challenging part of this construction is to encode the mechanism for mining rules in a *linear* way. Here, (19) requires the antecedent and consequent to be non-overlapping. (20) helps to define a_i: if item j is in the antecedent (so $\alpha_j = 1$), and observation i does not contain item j (so that $U_{ij} = 0$), then the a_i must be set to 0. Since there is a constraint like this for each j, then if there is any j for which it holds, a_i must be 0. (21) states that when observation i includes a, then a_i is 1. To see this, $\mathbf{U}_{i\cdot}^T \boldsymbol{\alpha}$ is the number of items contained in both a and in observation i, whereas $\mathbf{1}^T \boldsymbol{\alpha}$ is the number of items in a. So whenever observation i includes all of a, then the difference between these terms is 0, and a_i is forced to be 1. (22) and (23) for the consequent are analogous to the previous two constraints for the antecedent. (24)-(25) define b_i correctly: whenever a_i and c_i are 1, then b_i must be 1. If either of them are 0, then b_i must also be 0.

Integrality is not needed for these variables, which is better from a computational perspective; they are automatically binary. For the a_i's, they are restricted to be within $[0,1]$ by (26), and because of (20) and (21) and the fact that the U_{ij} and α_j are binary, we have that either $a_i \geq 1$ or $a_i \leq 0$. Thus, each a_i must be either 0 or 1. A similar argument holds for the γ_j's, and integrality for the b_i's then follows from (24)-(26).

Now that we have defined any rule $a \rightarrow c$ obeying (19)-(26) to be valid, we next use the definition to find valid rules.

4.2 General Rule Mining

We can iteratively trace out rules at or near the optimal frontier, and also a band around it, using the algorithm below. This is a form of dynamic programming, where we fix the supports of the antecedent (A_t) and consequent (C_t), and solve for the rules near the frontier (if there are any, given A_t and C_t). We iteratively increase A_t and C_t until we trace out the full frontier and a band around it. We start with minimum support thresholds $A_{\min}$ and $C_{\min}$. The increasing threshold B_{A_t,C_t} is the threshold for the support of b determining how close to the frontier we will collect rules from when we are at supports A_t for the antecedent and C_t for the consequent. We will mine a band around the frontier that is approximately η percent close to the frontier, where η is actually set by the user ($\eta = .95$ for us). Points on the frontier will have large values of $\tilde{B}$, which is the maximum support of rules at the point A_t, C_t. The algorithm is:

Initialize: $B_{A_{\min},C_{\min}} = 0$
For $A_t = A_{\min}$ to m
  For $C_t = C_{\min}$ to m

- Solve formulation (27) below to obtain $\tilde{B}_{A_t,C_t}$.
- Update the threshold for b as
  $B_{A_t,C_t} = \max\left(\eta \cdot \tilde{B}_{A_t,C_t}, \max_{A \leq A_t, C \leq C_t} B_{A,C}\right)$.
- Include Equation (28) in the list of constraints in (27) to prevent finding the same solution again.
- Repeat previous three steps until (27) yields no feasible solutions.

  End For
End For

The formulation for finding rules of maximum support with sparsity constraints on the size of the rules is:

$$\begin{aligned}
&\tilde{B}_{A_t,C_t} = \max_{\boldsymbol{\alpha},\boldsymbol{\gamma},\mathbf{a},\mathbf{c},\mathbf{b}} \quad \sum_{i=1}^{m} b_i \quad \text{s.t.} \\
&\sum_{i=1}^{m} a_i \geq A_t, \quad \sum_{i=1}^{m} c_i \geq C_t, \quad \sum_{i=1}^{m} b_i \geq B_{A_t,C_t} \\
&\sum_{j=1}^{N} \alpha_j \leq \Lambda, \quad \sum_{j=1}^{N} \gamma_j \leq \Gamma \\
&\quad (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \mathbf{a}, \mathbf{c}, \mathbf{b}) \text{ are } \textit{valid}, \text{ obeying (19)-(26).}
\end{aligned} \tag{27}$$

Each time we discover a rule near the frontier, we add a constraint to force the rule to be infeasible the next time we solve so we do not find it again. Denoting the rule we discovered at the previous iteration to have antecedent $\boldsymbol{\alpha}_{\text{prev}}$ and consequent $\boldsymbol{\beta}_{\text{prev}}$, we include the so-called *no-good* constraint:

$$\begin{aligned}
&\sum_{j:\alpha_{\text{prev},j}=0} \alpha_j + \sum_{j:\alpha_{\text{prev},j}=1} (1-\alpha_j) \\
&+ \sum_{j:\beta_{\text{prev},j}=0} \beta_j + \sum_{j:\beta_{\text{prev},j}=1} (1-\beta_j) \geq 1.
\end{aligned} \tag{28}$$

If vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_{\text{prev}}$ are the same, and $\boldsymbol{\beta}$ and $\boldsymbol{\beta}_{\text{prev}}$ are the same, then this constraint is violated. Thus, at least one term must be different between the previous and current rules.

4.3 Mining Rules for Building Binary Classifiers

Rule mining is much simpler when the consequents are either +1 or -1. We mine rules for each class separately to ensure that each class has enough rules for the rule-ordering step. Thus, each class has its own 2-dimensional frontier. Fixing the class eliminates one constraint from the partial order since $\sum_{i=1}^{m} c_i$ is a constant: it is the number of training examples in that class. The partial order defining the frontier for class +1 becomes: $a^{(1)} \rightarrow +1 \preceq a^{(2)} \rightarrow +1$ whenever $\text{supp}(a^{(1)} \rightarrow +1) \leq \text{supp}(a^{(2)} \rightarrow +1)$ and $\text{supp}(a^{(1)}) \geq \text{supp}(a^{(2)})$; similarly for class -1. To trace out a band around the frontier for class +1 (without loss of generality) the formulation (27) reduces to:

$$\tilde{B}_{A_t} = \max_{\mathbf{a},\boldsymbol{\alpha}} \quad \sum_{i:y_i=1} a_i \tag{29}$$
$$\text{s.t.} \quad \sum_{i=1}^{m} a_i \geq A_t, \quad \sum_{i:y_i=1} a_i \geq B_{A_t}, \quad \sum_{j=1}^{N} \alpha_j \leq \Lambda$$

and $\mathbf{a}$, $\boldsymbol{\alpha}$ are *valid*, which reduces just to Equations (20) and (21) along with $\alpha_j \in \{0,1\}\ \forall j$ and $0 \leq a_i \leq 1\ \forall i$. We loop over A_t from the minimum support threshold $A_{\min}$ to m when we are tracing out the frontier, choosing A_t to be either η times the support of the antecedent for the discovered rule, or the previous A_t, whichever is higher. Constraint (28) reduces to:

$$\textstyle\sum_{j:\alpha_{\text{prev},j}=0} \alpha_j + \sum_{j:\alpha_{\text{prev},j}=1} (1-\alpha_j) \geq 1.$$

As discussed earlier, we need only that the set of rules generated by the rule mining method is large enough to build an accurate predictive model.

We remark that the rule mining methods provided here mine conjunctive rules, and thus can be viewed as a type of optimal inductive logic programming. The methods can handle additional arbitrary linear constraints.

Another remark is that one could use this type of rule mining as part of an algorithm that mines rules during the optimization process for the rule list. For instance, one could replace our rule ordering method with a column-generation method for calculating possible rules. However, the rule generation process might be redundant (repetitive), leading to slowness of the overall strategy.

5 Strategic Computation

In this section we show three results that will ensure the consistency of the solution while assisting with computation. First we show that if we mine all high support rules, the set of pre-mined rules will contain the globally optimal solution. This means we do not need to mine low support rules, as they would not be in the optimal rule list anyway. Since the number of possible low support rules is usually enormous, it is beneficial to prove that we need not consider them at all.

The second and third results in this section provide bounds that are specialized to the objective in Section 3. These bounds can be used with callback functions, or in specialized (non-MIP) implementations. They allows us to add additional constraints throughout the computation to exclude large parts of the hypothesis space without exploring them. In follow-up work, they have shown to be particularly useful in excluding parts of the search space for specialized implementations (Angelino et al, 2018). The usefulness, and relative usefulness, of these bounds in practice can change dramatically between datasets. These bounds can become invalid if the user changes the objective or constraints within the formulation. For instance, if we change the constraints to force the rule list to be "falling" (as in work of Wang and Rudin, 2015; Chen and Rudin, 2018), the theorems are not valid. Thus, these bounds should not be included in a general formulation, only in specialized formulations when the bounds are valid.

5.1 On Optimality of Rule Lists from Mined Rules

In this section, we show that finding a globally optimal rule list (among selection and permutation of *all* possible rules) is equivalent, under weak conditions, to the rule list our method finds when our pre-mined rule set is the set of high support rules. This means we do not need to mine any low-support rules, as they would never be in an optimal rule list. The globally optimal rule list would be obtained by using every possible rule (every conjunction of conditions) and applying the Step 2 formulation in Section 3;

the number of conjunctions increases exponentially in the number of features, and thus the total number of total (but mostly low support) rules can be extremely large. We will show that when the regularization parameter on the number of rules is C (that is, when the user is willing to sacrifice accuracy on C training examples to obtain one fewer rule in the list), we need only mine all rules with support at least C in Step 1, and the globally optimal solution will be attained.

Note that mining all rules with a minimum support threshold is an extremely well-studied enumeration problem in data mining, for which one does not need an algorithm like that of Section 4.3. Our rule mining algorithm would be useful in cases where the number of high support rules is too large to use in the rule ordering algorithm, or in cases where one wants to find optimally interesting individual rules.

Let us start with notation similar to that in Section 1, allowing this result to be self-contained. Again, the rule list is $f = (L, \{(a_l, \gamma_l)\}_{l=1}^{L})$ with number of rules $L \in \mathbb{Z}^+$, $a_l(\cdot) \in B_x(\cdot)$ for $l = 1 \dots L-1$, where $B_x(\cdot)$ denotes the space of boolean functions of feature space X, and the default ELSE rule is $a_L(x) = \text{True}\ \forall x$. Again, γ_l is the predicted outcome (either 0 or 1) for rule l. Notation $f(x)$ means the prediction made by the first rule that x satisfies, $f(x) = \gamma_l$, where $l = \min l'$ s.t. $c_{l'}(x) = \text{True}$. Some new notation is as follows: the number of rules in the list will be denoted for clarity as $\#\text{Rul}(f) := L$. The full space of rule lists that use any possible combination of rules is $\mathcal{F}$. We also define $\mathcal{F}_C^S$ as the set of rule lists whose rules have support at least C:

$$\mathcal{F}_C^S := \{f : \min_l \text{supp}^S(a_l) \geq C\},$$

where $S = \{(x_i, y_i)\}_i$, where $\text{supp}^S(a)$ is the number of observations in S where condition a is satisfied. Note $\mathcal{F}_0^S = \mathcal{F}$, which means we just use all rules. We will use the following objective, which is a simplified variation of that in Section 3, which uses minimization rather than maximization:

$$R_S(f) = \sum_i 1_{[y_i \neq f(x_i)]} + C\ \#\text{Rul}(f).$$

The main result of this subsection is as follows:

Theorem 1 *(Minimum Support Bound)* $\operatorname{argmin}_{f \in \mathcal{F}} R_S(f) = \operatorname{argmin}_{f \in \mathcal{F}_C^S} R_S(f)$.

This theorem states that the set of functions on the left (which are computationally difficult to compute in practice) are the same as the set of functions on the right (which consist of functions that we are often computing in practice). It states that we need only mine rules of support of at least C; if we solve for a minimizer of R_S on pre-mined high support rules, it is a global minimizer of R_S across all possible rule sets. This provides strong computational motivation for pre-mining rules, since Theorem 1 states weak, checkable, conditions under which the two-step process using pre-mined rules yields an optimal solution.

Proof Again assume for ease of notation that rules in the list are ordered so that rules are listed from top to bottom as $a_1, a_2, ..., a_L$. Choose any $f \in \mathcal{F}$, $f = (L, \{(a_l, \gamma_l)\}_{l=1}^{L})$. We will show that if any of the a_l's have support less than C on data S, then f cannot be an optimal solution, $f \notin \operatorname{argmin}_{f \in \mathcal{F}} R_S(f)$. This is sufficient, since we will have shown that all optimizers of $R_S(f)$ are also in $\mathcal{F}_C^S$. So assume rule r has $\text{supp}^S(a_r) < C$, and we will then show $f \notin \operatorname{argmin}_{f \in \mathcal{F}} R_S(f)$. Define $f^{\setminus r}$ which has rule r removed:

$$f^{\setminus r} = (L-1, \{(a_l, \gamma_l)\}_{l=1\dots L, l \neq r}).$$

We will show that $f^{\setminus r}$ has a strictly lower value of the objective than f does, which will prove $f \notin \operatorname{argmin}_{f \in \mathcal{F}} R_S(f)$.

Define $\#\text{Captr}^S(f, r)$ as the number of observations classified by rule r ("captured" by rule r). That is,

$$\#\text{Captr}^S(f, r) = \sum_{i=1}^{n} 1_{[r = \text{argmin}\,\{l' \text{ s.t. } a_{l'}(x_i) = \text{True}\}]}.$$

First, we observe that for rule list f, rule r must classify less than C observations. This is because its support on the dataset (and thus the number of rules it applies to) is less than C. In the worst case, all

of these observations were classified correctly by r and classified incorrectly by the rules in $f^{\setminus r}$. Thus,

$$\begin{aligned} R_S(f^{\setminus r}) &= \sum_i 1_{[y_i \neq f^{\setminus r}(x_i)]} + C\#\text{Rul}(f^{\setminus r}) \\ &\leq \sum_i 1_{[y_i \neq f(x_i)]} + \#\text{Captr}^S(f,r) + C\left[\#\text{Rul}(f) - 1\right] \\ &= \sum_i 1_{[y_i \neq f(x_i)]} + \#\text{Captr}^S(f,r) + C\#\text{Rul}(f) - C \\ &= R_S(f) + \#\text{Captr}^S(f,r) - C. \end{aligned} \tag{30}$$

We always have, by definition, $\#\text{Captr}^S(f,r) \leq \text{supp}^S(a_r)$, and by assumption, $\text{supp}^S(a_r) < C$, so $\#\text{Captr}^S(f,r) - C < 0$. Continuing from (30),

$$R_S(f^{\setminus r}) \leq R_S(f) + \#\text{Captr}^S(f,r) - C < R_S(f).$$

Thus $f \notin \operatorname{argmin}_{f\in\mathcal{F}} R_S(f)$. This shows every optimal solution must have support at least C for all rules,

$$\operatorname*{argmin}_{f\in\mathcal{F}} R_S(f) \subseteq \operatorname*{argmin}_{f\in\mathcal{F}_C^S} R_S(f),$$

and because $\mathcal{F}_C^S \subset \mathcal{F}$, $\operatorname{argmin}_{f\in\mathcal{F}} R_S(f) = \operatorname{argmin}_{f\in\mathcal{F}_C^S} R_S(f)$.

5.2 Lower-Bound Constraints and Constraints on the Maximum Length of a Rule List

The constraints in this section will eliminate large parts of the hypothesis space that we can prove do not contain the optimal solution. We use these in practice for warm-starting computation manually when we stop it to examine the solution, and it often makes the search more efficient. These bounds can also be incorporated using call-back functions, or they can be directly incorporated into a custom branch-and-bound strategy, as done by Angelino et al (2018); Chen and Rudin (2018), who extend these bounds.

The intuition is as follows. We examine the first few rules (the "prefix") of our current list. Suppose we imagine we have a rule list starting with the same prefix, that classified everything below the prefix perfectly with only one more rule. If this imagined rule list is worse than that of another rule list we have seen, then even in the best possible circumstance, this prefix cannot achieve the best possible score. Thus, we no longer need to evaluate any rule list starting with this prefix.

We use the same notation as in the previous subsection. Define $\text{Captr}(f,l,i)$ as 1 if observation i is captured by the rule from model f with position l, and 0 otherwise.

Theorem 2 *(Objective Lower Bound) Let $v' = R_S(f_{ref})$ for model f_{ref}. (Here the reference model f_{ref} could be the best model we have found so far.) Let f^{curr} be our current model, $f^{\text{curr}} = (L, \{a_l, \gamma_l\}_{l=1}^L)$ where a_l is the l^{th} antecedent in the list. Assume that for f^{curr} the γ_l's are the majority vote for rule l so they are optimal with respect to accuracy.*

If for some $\theta \in \{1,...,L\}$, it is true that

$$v' < \sum_{i=1}^{m}\sum_{l=1}^{\theta} 1_{[y_i \neq \gamma_l]}\text{Captr}(f^{\text{curr}}, l, i) + C\theta,$$

then for $f^* \in \operatorname{argmin}_f R_S(f)$, $f^* = (L^*, \{a_l^*, \gamma_l^*\}_{l=1}^{L^*})$ we know

$$[a_1^*, ..., a_l^*, ..., a_\theta^*] \neq [a_1, .., a_l, .., a_\theta].$$

That is, the best possible model does not start with the collection and order of rules in our current solution.

Proof Consider the risk of model f^{curr}.

$$\begin{aligned} R_S(f^{\text{curr}}) &= \sum_i 1_{[y_i \neq f^{\text{curr}}(x_i)]} + C\#\text{Rul}(f^{\text{curr}}) \\ &= \sum_i \sum_{l=1}^{L} 1_{[y_i \neq \gamma_l]} \text{Captr}(f^{\text{curr}}, l, i) + CL \\ &\geq \sum_i \sum_{l=1}^{\theta} 1_{[y_i \neq \gamma_l]} \text{Captr}(f^{\text{curr}}, l, i) + C\theta > v'. \end{aligned}$$

The third line comes from calculating errors only from within the top θ rules, and the second term holds because $\theta \leq L$. The relationship to v' is the assumption of the theorem.

This inequality shows that $f^{\text{curr}} \notin \text{argmin}_{f'} R_S(f')$. This inequality also holds for any $\tilde{f}$ starting with f^{curr}'s prefix $a_1, ..., a_\theta$ and optimal right hand sides $\gamma_1, ..., \gamma_l$. That is,

$$R_S(\tilde{f}) \geq \sum_i \sum_{l=1}^{\theta} 1_{[y_i \neq \gamma_l]} \text{Captr}(\tilde{f}, l, i) + C\theta = \sum_i \sum_{l=1}^{\theta} 1_{[y_i \neq \gamma_l]} \text{Captr}(f^{\text{curr}}, l, i) + C\theta > v'.$$

This means that no model with prefix $a_1, ..., a_\theta$ and optimal right hand sides $\gamma_1, ..., \gamma_l$ could be a minimizer of R_S.

To clarify the indexing, height and position in the list are inverses. If a_l is the l^{th} rule in the list, its index within the collection of rules can be calculated from the inverse of the height function, $r = h^{-1}(R+1-l)$. Let us define our current model's δ variables as δ^{curr}, where $\delta^{\text{curr}}_{rj} = 1$ whenever rule r is in height j. That is, $\delta^{\text{curr}}_{rj} = 1$ for (r, j) pairs $(h^{-1}(R), R)$, $(h^{-1}(R-1), R-1)$,..., $(h^{-1}(R+1-l), R+1-l)$,..., $(h^{-1}(R+1-\theta), R+1-\theta)$, etc. Let us denote the first θ of these pairs for our current solution as (r^{curr}_1, R), $(r^{\text{curr}}_2, R-1)$, $(r^{\text{curr}}_3, R-2)$, $(r^{\text{curr}}_4, R-3)$,...,$(r^{\text{curr}}_\theta, R+1-\theta)$.

In order to prevent the algorithm from looking at this prefix again, or its extensions, we must place a constraint saying that least one of the first elements of any discovered prefix must be different than those of the current prefix. That constraint is provided in the following corollary.

Corollary 2 *(Prevent Identical Solutions) If conditions of Theorem 2 are satisfied, that is, for some* $\theta = \{1, ..., L\}$,

$$v' < \sum_{i=1}^{m} \sum_{l=1}^{\theta} 1_{[y_i \neq \gamma_l]} \text{Captr}(f^{\text{curr}}, l, i) + C\theta,$$

then the following constraint is satisfied for the optimal solution f^**:*

$$\delta_{r^{\text{curr}}_1, R} + \delta_{r^{\text{curr}}_2, R-1} + \cdots + \delta_{r^{\text{curr}}_l, R+1-l} + \cdots + \delta_{r^{\text{curr}}_\theta, R+1-\theta} \leq \theta - 1. \tag{31}$$

Proof The constraint (31) in the corollary means that at least one of the θ variables $\delta_{r^{\text{curr}}_l, R+1-l}$ is 0. Recall that δ_{rj} is a binary variable that is 1 when rule r is in rank j. If all of the δ's in the sum were 1, it would imply through Equation (6) that the first θ rules agree exactly with those in f^{curr} which we know are not optimal by Theorem 2.

Thus, whenever our current solution satisfies the condition of Theorem 2, we can add the cut in Corollary 2 to the set of constraints within the formulation. This prevents us from having to further consider rule lists with the same provably bad prefix as in the current solution.

Cuts of this form with small θ are generally more useful than cuts with large θ because they remove larger parts of the hypothesis space. For instance, a cut with $\theta = 2$ would remove all rule lists with a particular prefix of 2 rules, whereas a cut with $\theta = 7$ removes only rule lists starting with the same 7 rules.

Many cuts of this form could be potentially added (automatically or manually) as the optimization procedure proceeds.

Rank	Rule	Train		Test	
		Captured	Correct	Captured	Correct
1	$x1\ x2\ x4 \rightarrow$ No Win	26	14	15	3
2	$x1\ x7\ x8 \rightarrow$ No Win	22	8	15	5

Table 4 Bad rule list prefix for Theorem 2.

5.2.1 Example for Theorem 2

Consider the rule list prefix in Table 4. This prefix is bad. The first rule captures 26 training observations, and gets only 14 correct, whereas the second rule captures 22 observations and gets only 8 correct. We calculate v' from the theorem, using the rule list we found from optimization as the reference function, which has 0 errors and 9 rules.

$$v' = C \times \text{num. rules in list} = \frac{1}{R} \times 9 = \frac{1}{13} \times 9 = 0.692.$$

The right side of the theorem is calculated from the rule list as:

$$(26 - 14) + (22 - 8) + C \times \theta = 26 + \frac{1}{13} \times 2 = 26.154.$$

Since the first quantity is less than the second, the condition of the theorem is satisfied, and we can remove any rule list starting with the bad prefix in Table 4.

The last result is a bound on the maximum number of rules in the optimal solution. This bound can be updated within the optimization procedure as new reference solutions are obtained. This bound was also used within the extension of this work, Angelino et al (2018).

Theorem 3 *(Length Bound) Let us say without loss of generality that the positive class is the minority class. Let f^* be the optimal classifier, $f^* \in \text{argmin}_f R_S(f)$, and denote its number of rules by L^*. Let f_{ref} be any reference function. Then*

$$L^* \leq \frac{\sum_i 1_{[y_i=1]}}{C}, \text{ and } L^* \leq \frac{R_S(f_{\text{ref}})}{C}.$$

The first inequality in the bound can be used before a good reference solution is found. As better reference solutions are found, the second inequality can be used.

Proof Consider the model that is the empty set of rules, denoted by $\emptyset$. We have $\sum_{i=1}^{m} 1_{[y_i=1]} = R_S(\emptyset)$ by definition of R_S because $\emptyset$ misclassifies all the positive examples and the penalty for the number of rules vanishes because there are no rules. Also, by definition of f^* being the optimal classifier, $R_S(f^*) \leq R_S(f)$ for all f including $\emptyset$.

$$\sum_{i=1}^{m} 1_{[y_i=1]} = R_S(\emptyset) \geq R_S(f^*) = \sum_i 1_{[y_i \neq f^*(x_i)]} + CL^* \geq CL^*. \tag{32}$$

Rearranging yields the first result. Similarly

$$R_S(f_{\text{ref}}) \geq R_S(f^*) = \sum_i 1_{[y_i \neq f^*(x_i)]} + CL^* \geq CL^*. \tag{33}$$

To put a constraint into the bound stating that the length of the list is at most L_{bound}, we would use the inequality $\tilde{\pi} \geq R - L_{\text{bound}} + 1$ within the formulation.

6 Evaluation

We present experimental results showing: 1) that the approach to rule mining and ordering described above ("Ordered Rule Lists" - ORL) can make predictions that are as accurate as other machine learning methods, and 2) it can find correct patterns in data that other algorithms cannot.

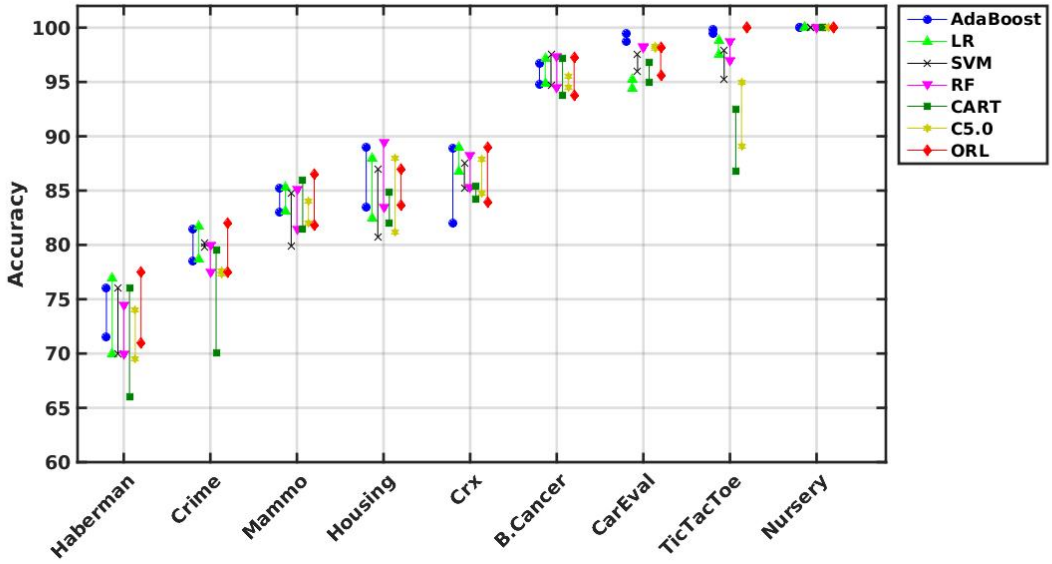


Fig. 1 Test accuracies for all algorithms for different datasets. Algorithms behave similarly across datasets.

6.1 Data Details and Accuracy Tables

Figure 1 shows a summary of the accuracy results. It shows the accuracy of several algorithms on various healthcare and sociological datasets (Bache and Lichman, 2013), and a criminological dataset, ("Crime" Cusick et al, 2010) averaged over three test folds, showing similar accuracy performance for all algorithms on each of the datasets. Table 10 in Appendix A contains more detail about experiments, specifically, the numerical values for accuracy for all algorithms, and all pairwise hypothesis tests.

Table 5 has more details about the data and number of rules generated by ORL for these datasets, and implementation details for all methods. It also shows the size of the decision trees produced by CART and C5.0. Algorithms we compare with include boosted decision trees – AdaBoost (Freund and Schapire, 1997), logistic regression – LR, support vector machines with RBF kernels – SVM (Vapnik, 1998), random forests – RF (Breiman, 2001), CART, and C5.0. Most of these methods have difficulty handling user-defined constraints. For all baseline methods, we used the Weka (Hall et al, 2009) software package on a Mac-Book Pro with 2.53 GHz Intel i5 core processor. Several of the algorithms have no parameters to tune (AdaBoost, logistic regression). Heuristics were used to set some of the parameters for SVM (C=1, RBF kernel parameter = 1/#Features), though we report results from SVM for many possible parameter values for all datasets in Table 11 in Appendix A. Default parameters were used for random forests.

ORL was implemented in Ampl with the Gurobi solver and the experiments were conducted in Linux environment with a 2.66GHz Intel Xeon processor. All rule generation routines were solved to optimality. For rule ranking, we set time limits based on the number of rules, shown in Table 6. The programs sometimes solved to optimality well before the time limit was reached (e.g., for Tic Tac Toe and Nursery). For the purpose of comparison with other methods we ran the solver many times, which necessitated time limits, but in practice one would only run the solver a few times (possibly revising constraints and the objective in between runs), and such time limits would not be necessary. For ORL, we used a single setting, where C_1 was set at $1/R$, and C was chosen to be a small constant less than C_1 to favor horizontal sparsity over vertical sparsity. From a practical perspective, there was not a clear "winning" algorithm, and ORL was on par with other algorithms.

6.2 Case Study Showing Problems with Greedy Methods

We now show a case study illustrating the problems with greedy methods like CART that are sidestepped by ORL. Consider the Tic Tac Toe dataset (Bache and Lichman, 2013) discussed earlier, where, given a finished tic tac toe board, the goal is to determine whether the X player has won the game. This is simple for a human, who would look for three X's in a row. The question is whether the computer can learn

	n	d	**#Rules**
B.Cancer	683	27	397±38
CarEval	1728	21	250±13
Crime	558	49	273±32
Crx	653	60	753±89
Haberman	306	10	31±1
Housing	506	49	1288±306
Mammo	830	25	90±4
Nursery	3240	27	45±4
TicTacToe	958	27	109±7

	CART	**C5.0**	**ORL**
B.Cancer	5	5.3	12.7
CarEval	14	27.7	12.7
Crime1	11.3	16.3	21
Crime2	8.3	16.3	9.3
SPECT	6.3	7.7	18
Haber	4.7	3.7	4.3
Mammo	4	8.7	12.3
MONK2	14.7	29.3	19.7
TicTacToe	21	34.7	9
Titanic	3.7	5	3.3
Votes	3	4.7	7.3
Average	8.7	14.5	11.8
Standard Dev.	5.8	11.3	5.9

Table 5 Characteristics of datasets used in experiments. Left: Number of observations (n), number of items (d) and average number of rules generated for each dataset. Right: Average number of leaves for CART and C5.0, and average length of decision list for ORL. CART's trees tend to be very short, but accuracy is often sacrificed.

Dataset	Avg rule generation time (sec.)	Rule ordering limit (sec.)
TicTacToe	4,612	7,200
Haberman	53	7,200
Car Eval	10,774	43,200
Mammo	623	54,000
Housing	744	54,000
B. Cancer	554	36,000
Crime	6,144	72,000
Crx	957	43,200
Nursery	312	7,200

Table 6 Time limits placed on solver.

this pattern from examples. As it turns out, none of the state-of-the-art classifiers detected the pattern – this can be seen directly from the accuracy results in Figure 1, since none of the other algorithms have perfect test results. Trees from CART and C5.0 are presented in Figures 5 and 7 in Appendix B, which are not particularly interpretable. Even as we varied C5.0 and CART parameters across their full ranges, they were not able to detect the pattern for any parameter setting we tried; Figures 6 and 8 in Appendix B display this explicitly, where we considered the full range of their "confidence factor" parameter values and were still not able to get CART nor C5.0 to produce the right answer. These results show the danger of using greedy methods – they can fail to find important patterns in data. Both CART and C5.0 are theoretically capable of producing the right answer, but to get that answer it would require so much parameter tuning that an extra algorithm may be needed to conduct that tuning.

The model from one of the ORL folds is in Figure 2, stating that three X's in a row in one of eight different ways means that X wins, and otherwise X did not win. Models for other folds are Tables 12 and 13 in Appendix C, and the example in Table 1 was computed by ORL on the full dataset. All of the folds and full dataset training supplied a completely correct model with no tuning. All methods had exactly the same input, but only ORL was able to consistently learn what constitutes winning.

One way the user can customize ORL is by adjusting its tradeoff parameter C to produce models at varying levels of sparsity. Several models with various choices for C are shown in Table 7 for the Haberman Survival dataset (Bache and Lichman, 2013), where the goal is to predict 5 year survival after a breast cancer diagnosis at the University of Chicago's Billings Hospital. The age parameter is the age of the patient, the number of nodes is the number of positive axillary nodes detected, and the year is the year of the patients' operation. The models in Table 7 are of sizes 8, 6, and 4 respectively. C can be set using cross validation also, but since C is meaningful as the amount of training accuracy sacrificed for one fewer term in the list, it is often useful to set it to obtain the desired level of sparsity. Assuming the user is interested in sparse, interpretable models, they will most likely set C large enough to generalize to the test set. Results for varying C_1 are in Appendix D.

We used ORL to design a violence prediction model. The data are from a large-scale U.S. Department of Justice Statistics study of youth raised in out-of-home care (Cusick et al, 2010), involving survey and arrest data from several hundred youths. Our goal is to predict whether a youth will commit a violent crime before age 21, using data collected prior to the youth's 18th birthday, including detailed criminal

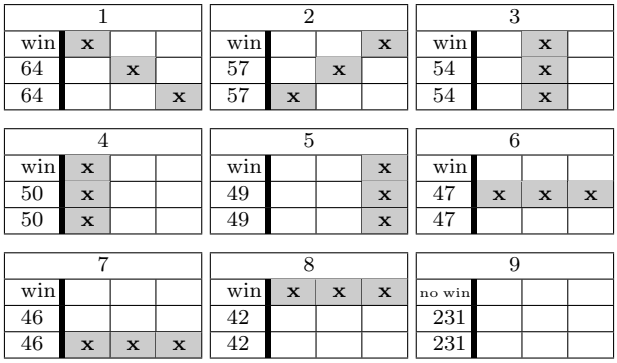


Fig. 2 Tic Tac Toe ordered rules for Split 1. The first rule is denoted by the tic tac toe board on the upper left, indicating that if there are three X's in the three gray squares of the board, then the X player wins, and that is true regardless of what else is on the board. Else we move to the second rule in the upper middle, where we check for three x's in a row on the other diagonal (and ignore anything else on the board), in which case the X player wins. The third rule is on the upper right, and so on. If there are not three X's in a row on the board, then none of the 8 rules are satisfied, in which case the default (labeled "9") on the lower right states that the X player will not win. The numbers on the left of each table (e.g., 64, 57, 54, etc.) indicate how many observations are captured by the rule, and of these observations, how often the rule made a correct prediction. (The numbers in each pair are equal, meaning each rule made a correct prediction for every observation it captured.)

	Rule			
1	Age <40	$\Rightarrow$	≥ 5	(23/27)
2	Number of Nodes (1 to 9)	$\Rightarrow$	≥ 5	(47/68)
3	Year >1965	$\Rightarrow$	≥ 5	(27/29)
4	No Nodes	$\Rightarrow$	≥ 5	(49/60)
5	Large Number of Nodes (10 to 19)	$\Rightarrow$	<5	(7/9)
6	$1962 \leq$ Year ≤ 1963	$\Rightarrow$	<5	(3/4)
7	Age 60-69	$\Rightarrow$	≥ 5	(3/3)
8	Default	$\Rightarrow$	<5	(4/4)

	Rule			
1	(Year <1960) **and** Large Number of Nodes (10 to 19)	$\Rightarrow$	<5	(4/6)
2	Age 50-59 **and** (Year >1965)	$\Rightarrow$	≥ 5	(14/17)
3	Age 60-69	$\Rightarrow$	≥ 5	(30/38)
4	Age 50-59 **and** Large Number of Nodes (10 to 19)	$\Rightarrow$	<5	(2/3)
5	Very Large Number of Nodes (20 to 29)	$\Rightarrow$	<5	(5/6)
6	Default	$\Rightarrow$	≥ 5	(104/134)

	Rule			
1	(Year <1960) **and** Large Number of Nodes (10 to 19)	$\Rightarrow$	<5	(4/6)
2	$60 \leq$Age ≤ 69	$\Rightarrow$	≥ 5	(30/38)
3	Very Large Number of Nodes (20 to 29)	$\Rightarrow$	<5	(5/7)
4	Default	$\Rightarrow$	≥ 5	(118/153)

Table 7 *Top:* Model from Haberman survival dataset, using regularization parameter $C = \frac{1}{\#\ rules}$. Train/test accuracy = 0.7990 / 0.7745. No training accuracy is sacrificed for sparsity. *Middle:* Model from Haberman survival dataset, using regularization parameter $C = \frac{1}{4}$. Train/test accuracy = 0.7794 / 0.7549. The increase in regularization reduced training accuracy slightly in order to make the model sparser than that of the one above. *Bottom:* Model from Haberman survival dataset, using regularization parameter $C = 1$. Train/Test Accuracy = 0.7696 / 0.7353. The increase in regularization reduced training accuracy slightly in order to make the model sparser than that of either of the two above models.

history and demographic information. Table 8 shows ORL's learned model on the dataset, trained on 2/3 of the data. The model states that being female, attending school, and being drug-free are indicators of not being involved in future crime, whereas prior offenses are indicators of future crime. The model also indicates something non-intuitive, which is that being close to one's father increases the chance for violent crime. Note that these youths are raised in out-of-home care, so it is not clear that their parents would be ideal role-models. It is also non-intuitive that neglected youths would be less likely to commit a violent crime; however, recall that these youths were raised in out-of-home care, so there had to be at least one reason for their having been raised away from home (e.g., neglect), so this result may have been a function of the original conditioning of the dataset. "Entry over 12" means that the youth entered out of home care after 12 years of age.

	Rule			% Correct
1	Female **and** No Children **and** In school	⇒	No crime	(127/135)
2	No alcohol or drug dependency **and** In school **and** Neglected	⇒	No crime	(70/87)
3	Entered foster care over 12	⇒	No Crime	(57/74)
4	Involved in violent crime **and** Sold drugs **and** Close to father	⇒	Crime	(8/11)
5	Male **and** Involved in violent crime **and** In school	⇒	Crime	(6/12)
6	Default	⇒	No crime	(39/53)

Table 8 Set of rules for the crime dataset. The model states that if the individual is female with no children then it is unlikely she will commit a crime, and shows that in the database, 127 out of 135 individuals matching these criteria did not commit a crime. If the first rule's conditions are not satisfied, the second rule's conditions are checked, etc.

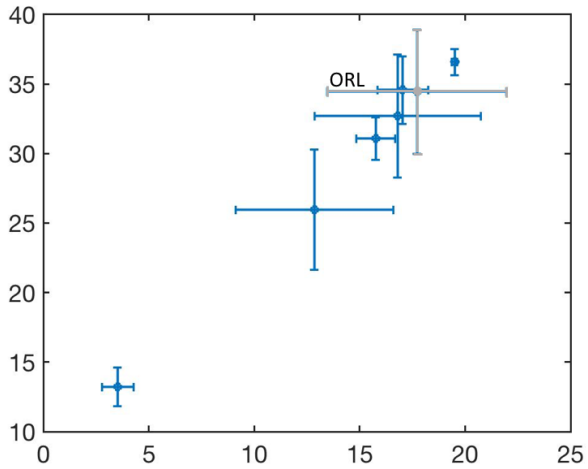


Fig. 3 G-means versus F-scores on the test set for the violent crime dataset (means and standard deviation computed across folds). Each marker represents an algorithm (AdaBoost, logistic regression, support vector machines, random forests, CART, C5.0, ORL). Most algorithms perform similarly. ORL is marked in gray.

Figure 3 (and Table 15 in the appendix) shows the empirical results for the violent crime dataset. This dataset is imbalanced, which means accuracy is not necessarily a useful measure of success, so we report the F-score and G-means as evaluation measures in addition to the accuracy results in Figure 1. The F-score is the harmonic mean of recall and precision,[1] and G-means is the geometric mean of sensitivity and specificity. The results for most of the algorithms again is very similar in terms of F-Score and G-means.

As a side note, the ability to handle multiple constraints is not explored in our experiments. This is because no easy comparison can be made to methods, such as decision trees, that may never produce feasible models. We refer to experiments in other works that show that C4.5, C5.0R, C5.0T and CART tend to produce trivial models on imbalanced datasets; in particular, Goh and Rudin (2014) show on 37 imbalanced datasets that C4.5 and C5.0 often tended to produce trivial models that predict only one class, Zeng et al (2017) provide experiments showing that C5.0R, C5.0T were unable to produce sparse models on real-world criminology datasets, and that CART, C5.0R, and C5.0T produced only trivial models (predicting all one class) on some imbalanced criminology prediction problems. It may be possible to tune parameters further to achieve non-trivial models, but that could require a huge number of trials, and may not yield feasible models. The challenge of handling multiple constraints arises in linear models (as discussed by Ustun and Rudin, 2016, 2017), as well as other model forms, and for many other types of algorithms.

[1] The recall or sensitivity of a classifier is the true positive rate, the precision is the fraction of predicted positives that are true positives, the specificity is the true negative rate:

$$\text{recall} = \text{sensitivity} = \frac{\sum_i 1_{(y_i=f(x_i))\&(y_i=1)}}{\sum_i 1_{y_i=1}}, \quad \text{precision} = \frac{\sum_i 1_{(y_i=f(x_i))\&(y_i=1)}}{\sum_i 1_{f(x_i)=1}}, \quad \text{specificity} = \frac{\sum_i 1_{(y_i=f(x_i))\&(y_i=-1)}}{\sum_i 1_{y_i=-1}}.$$

	Positive Observations								
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
p_1	0	0	0	0	0	0	0	**1**	**1**
p_2	0	0	0	0	0	0	**1**	**1**	**1**
p_3	0	0	0	**1**	0	0	**1**	**1**	**1**
p_4	0	0	0	**1**	0	**1**	**1**	0	**1**
p_5	0	0	0	**1**	**1**	0	0	**1**	**1**
p_6	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**
p_7	0	0	**1**	0	0	0	**1**	**1**	**1**
p_8	0	0	**1**	0	**1**	0	**1**	**1**	**1**
p_9	0	0	**1**	0	**1**	**1**	**1**	**1**	**1**
p_{10}	0	0	**1**	**1**	0	**1**	0	0	**1**

	Negative Observations								
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
n_1	0	0	0	**1**	**1**	**1**	**1**	**1**	0
n_2	**1**	**1**	0	0	0	0	0	**1**	0
n_3	**1**	**1**	0	0	0	**1**	0	0	0
n_4	**1**	**1**	0	0	**1**	0	**1**	**1**	0
n_5	**1**	**1**	0	**1**	0	**1**	0	0	0
n_6	**1**	**1**	0	**1**	**1**	0	0	0	0
n_7	**1**	**1**	0	**1**	**1**	**1**	0	0	0
n_8	**1**	**1**	**1**	0	0	0	**1**	0	0
n_9	**1**	**1**	**1**	0	0	**1**	0	0	0
n_{10}	**1**	**1**	**1**	**1**	0	0	0	0	0

Table 9 Data for illustrating customization of objective and constraints.

In the next section, we demonstrate how ORL can be customized to the goals of the end user.

7 Customized Variations

By creating minor modifications of the formulation, we can customize the rule lists. To illustrate precisely how this works, we use a small dataset with 9 features f_1, f_2, ..., f_9 for which we can see how changes in the objective and constraints can alter the results of the algorithm.

Let us discuss data generation. For observations of the positive class, we generated each feature from a Bernoulli distribution with a different success probability, set at 0.1 increments $p_+(f_1 = 1) = 0.1, p_+(f_2 = 1) = 0.2, \ldots, p_+(f_9 = 1) = 0.9$. The feature values for observations of the negative class were generated similarly but the order of success probabilities was reversed; that is $p_-(f_1 = 1) = 0.9, p_-(f_2 = 1) = 0.8, \ldots, p_-(f_9 = 1) = 0.1$. This generated 10 positive (p_1 through p_{10}) and 10 negative (n_1 through n_{10}) observations shown in Table 9. We define ten positive rules $p_1 \Rightarrow 1$, $p_2 \Rightarrow 1$, $\ldots$, $p_{10} \Rightarrow 1$ and negative rules $n_1 \Rightarrow 0$, $n_2 \Rightarrow 0$, $\ldots$, $n_{10} \Rightarrow 0$ (e.g., $p_1 \Rightarrow 1$ is the rule "$f_8 = 1$ and $f_9 = 1 \Rightarrow$ predict 1", as shown in the top row of Table 9; we also write this rule as $f_8\ f_9 \Rightarrow 1$). Including the two default rules, there are a total of 22 rules in this simulation.

7.1 Favoring the use of certain features.

We can modify the algorithm so that the rule list will not use a particular feature v unless there is a gain in accuracy of at least 10 observations from using that feature. To do this, set parameter $C_{\text{Variable}} = 10$. Then, create a new parameter ρ which is a vector of size R. We set each entry ρ_r to be 1 if rule r uses feature v and 0 otherwise. Then, we add the following term to the objective (1):

$$-C_{\text{Variable}} V,$$

where the MIP will set the new variable $V \in \{0, 1\}$ to be 1 if any of the rules in the list use feature v. To do this, we add the following constraint to the MIP:

$$\beta_r + \rho_r - 1 \leq V.$$

This will force V to be 1 only when β_r and ρ_r are 1. V will be forced to 0 otherwise, from the term we added to the objective, and the fact that we are maximizing.

The "original" ranked rule list (with $C_{\text{Variable}} = 0$) is as follows (with other parameters C=C_1=$1/R$):

$$\begin{aligned} f_8\ f_9 &\Rightarrow 1 \\ f_4\ f_6\ f_7\ f_9 &\Rightarrow 1 \\ f_3\ f_4\ f_6\ f_9 &\Rightarrow 1 \\ \emptyset &\Rightarrow 0. \end{aligned}$$

This list achieves 100% accuracy, and this can be checked by hand from the data table. Since the non-default rules in this list all contain f_9, we discourage the use of that variable by setting $C_{\text{variable}} = 10$ to see what will happen. The resulting list does not have a rule with feature 9.

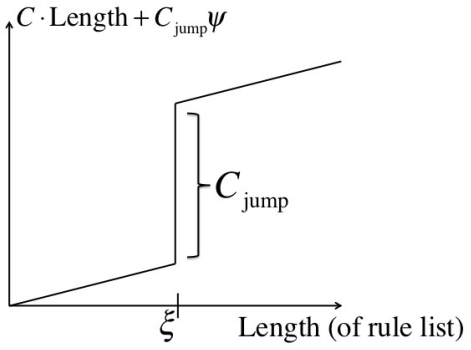


Fig. 4 Example of a nonlinear regularization for the number of rules in the list.

$$\begin{array}{rl} f_1\ f_2\ f_3\ f_7 & \Rightarrow\ 0 \\ f_1\ f_2\ f_8 & \Rightarrow\ 0 \\ f_1\ f_2\ f_4\ f_5 & \Rightarrow\ 0 \\ f_1\ f_2\ f_6 & \Rightarrow\ 0 \\ f_1\ f_2\ f_3\ f_4 & \Rightarrow\ 0 \\ \emptyset & \Rightarrow\ 1 \end{array}$$

The overall accuracy for this list is 95% (accuracy is 100% for the positive class, 90% for the negative class), and this can be checked by hand from the data table.

7.2 Regularizing a nonlinear function of the number of rules.

Currently our regularization for the number of rules is linear, but we may want a different kind of penalty. There are standard ways to encode nonlinear functions of integers into MIP. A simple example is where we would like a sharp increase in the penalty if the number of rules becomes above a threshold, say 3 rules. A diagram of the penalty is in Figure 4, where parameter ξ would be set to 3. To encode this as part of the MIP, we would include a new decision variable $\psi \in \{0,1\}$, which is 1 when the number of rules in the list (which is $R - \tilde{\pi} + 1$) obeys $R - \tilde{\pi} + 1 > \xi$. The constraint that defines ψ is

$$R - \tilde{\pi} + 1 - \xi \leq R\psi,$$

and the objective would be amended to include the term $C_{\text{jump}}\psi$. This means we would sacrifice the misclassification of at least C_{jump} observations to change the rule list from size ξ to $\xi + 1$.

Starting from the original rule list in Section 7.1, which is of size 4, we create an incentive for the MIP to reduce the size of the rule list by setting ξ to 3 to penalize lists longer than 3. Setting $C_{\text{jump}} = 10$ yields the list:

$$\begin{array}{rl} f_8\ f_9 & \Rightarrow\ 1 \\ f_3\ f_4\ f_6\ f_9 & \Rightarrow\ 1 \\ \emptyset & \Rightarrow\ 0. \end{array}$$

The overall accuracy for this list is 95% (accuracy is 90% for the positive class, 100% for the negative class).

7.3 Cost-sensitive learning.

A standard approach to cost sensitive learning is to weight the true positives differently than the true negatives. This allows the user to achieve a desired balance between sensitivity and specificity. Instead of creating the matrix $\mathbf{M}$, we create $\mathbf{M}^+$ and $\mathbf{M}^-$, defined elementwise by $M^+_{ir} = 1$ if rule r applies to i, $y_i = 1$, and rule r's right hand side is 1; $M^+_{ir} = 0$ otherwise. Similarly, $M^-_{ir} = 1$ if rule r applies to i, $y_i = -1$, and rule r's right hand side is -1; $M^-_{ir} = 0$ otherwise. We define C^+_{cost} to be the value of each

true positive, and C^-_{cost} to be the value of each true negative. For instance, we would set $C^+_{\text{cost}} = 2$ and $C^-_{\text{cost}} = 1$ if we wanted each correct positive to be worth twice as much as one true negative. Then the accuracy term in the objective is replaced with the following:

$$C^+_{\text{cost}} \sum_{i:y_i=1} \sum_r M^+_{ir} d_{ir} + C^-_{\text{cost}} \sum_{i:y_i=-1} \sum_r M^-_{ir} d_{ir},$$

the first term is the (weighted) number of correctly classified positive observations, and the second term is the (weighted) number of correctly classified negative observations.

Since the original rule list achieves 100% accuracy, we removed two rules, namely $p_1 \Rightarrow 1$ and $n_1 \Rightarrow 0$, to cause misclassifications. Thus, in this experiment we now have a total of 20 rules including the default rules. First, starting with equal costs of $C^+_{\text{cost}} = C^-_{\text{cost}} = 1$, the rule list is

$$\begin{aligned} f_7\ f_8\ f_9 &\Rightarrow 1 \\ f_4\ f_6\ f_7\ f_9 &\Rightarrow 1 \\ f_4\ f_5\ f_8\ f_9 &\Rightarrow 1 \\ f_3\ f_4\ f_6\ f_9 &\Rightarrow 1 \\ \emptyset &\Rightarrow 0 \end{aligned}$$

achieving an overall accuracy of 95% (accuracy is 90% for the positive class, 100% for the negative class). In order to force a higher accuracy on the positive class, we set $C^+_{\text{cost}} = 5$, which yielded the list

$$\begin{aligned} f_1\ f_2\ f_6 &\Rightarrow 0 \\ f_1\ f_2\ f_8 &\Rightarrow 0 \\ f_1\ f_2\ f_3\ f_7 &\Rightarrow 0 \\ f_1\ f_2\ f_4\ f_5 &\Rightarrow 0 \\ f_1\ f_2\ f_4\ f_5 &\Rightarrow 0 \\ f_1\ f_2\ f_3\ f_4 &\Rightarrow 0 \\ \emptyset &\Rightarrow 1 \end{aligned}$$

achieving the same overall accuracy of 95%, but now the accuracy on the positive class is 100% and it is 90% on the negative class.

As we have demonstrated, a major advantage of using MIP tools to construct models is the fact that the changes we demonstrated above are easy to make. However, when we do this, the customized cuts we designed earlier in Section 5 may no longer apply. It can be much easier to use the solver rather than to design a specialized method for each new set of constraints or objective the user might wish to try.

8 Related Work and Discussion

The goal of designing interpretable classifiers has existed since the beginning of artificial intelligence, (see reviews of Huysmans et al, 2011; Freitas, 2014; Vellido et al, 2012; Plate, 1999; Rüping, 2006). Models from the 1970's-1980's expert systems literature have the same form as a decision list (Leondes, 2002) so there is a historical precedent for this type of model. There is a body of work in psychology (Miller, 1956; Jennings et al, 1982) indicating that humans cannot handle very many cognitive entities at once (hence the need for sparse models in a wide variety of settings). The need for classifiers that can be adjusted and customized is also clear from the literature (see for instance, Freitas, 2014). There is also a case in the literature (see Freitas, 2014) that logical models are more useful than linear models in some settings.

Related to this paper is work on greedy methods besides decision trees, including decision list and associative classification methods, which are also top-down greedy (e.g., Rivest, 1987; Liu et al, 1998; Klivans and Servedio, 2006; Anthony, 2005; Long and Servedio, 2007; Marchand and Sokolova, 2005), inductive logic programming approaches (e.g., Muggleton and De Raedt, 1994) that locate logical rules but do not optimize how they are going to be used together in a predictive model, or other types of techniques that learn models for individual rules but do not learn how they should be combined (e.g., Malioutov and Varshney, 2013; Rudin et al, 2013; McCormick et al, 2012). Experiments of Wang and Rudin (2015); Yang et al (2017) found that the performance of inductive rule learner RIPPER (Cohen, 1995) was worse than that of other methods; rule learning methods generally do not aim to achieve or prove optimality. Reviews of associative classification techniques are provided by Thabtah (2007); Rückert (2008); Vanhoof and Depaire (2010). Bayesian tree methods (Chipman et al, 1998; Wu et al, 2007) find local posterior modes using top-down greedy methods like regular decision trees. Some rule-based models are similar to the Logical Analysis of Data (LAD) framework (Boros et al, 2000), though

the LAD model uses only rules that have confidence equal to one, so that even rules with confidence 0.99 are discarded. There are more modern techniques for creating more accurate individual rules (see, for instance, Su et al, 2016). There are methods that combine (possibly many) rules together to make predictions from rule ensembles, but these methods are not designed to be interpretable. Examples are the methods of Li et al (2001); Yin and Han (2003); Friedman and Popescu (2008); Meinshausen (2010).

There is a (mostly older) body of work on optimality in decision trees using dynamic programming (Bennett and Blue, 1996; Nijssen and Fromont, 2010; Farhangfar et al, 2008; Dobkin et al, 1996). Several of these works aim to solve the full problem and do not reduce the hypothesis space as we do with the pre-mining step. An exception is that of Nijssen and Fromont (2010, 2007), which can be used with pre-mined rules. This method requires any *leaf* of the tree to be fully pre-mined, including all of the conditions that led to that node. This is a different approach to this work, where only itemsets with a small number of conjunctions are mined. Nijssen and Fromont (2010) warn of issues of running out of memory when mining larger conjunctions, such as for the leaf nodes. The number of rules can grow exponentially with the number of conjunctions, so using smaller conjunctions helps with this (as well as with interpretability).

Our method for ordering rules can be used with any association rule mining method. Since the introduction of the Apriori method (Agrawal and Srikant, 1994), researchers have proposed many algorithms for frequent pattern mining, and rule learning. Reviews are given by Han et al (2007); Hipp et al (2000); Goethals (2003). There is a large body of literature on interestingness measures for rules, such as lift, conviction, Laplace, and gain, and reviews of interestingness measures are given by Tan and Kumar (2000); McGarry (2005); Geng and Hamilton (2006), and example methods are those of Simon et al (2011); Fawcett (2008). The focus of this work is not on ways to optimally combine rules, but instead on how to generate the rules.

A follow up paper to the present work is that of Angelino et al (2017, 2018), who build off the constraints here and develop a customized branch-and-bound solver for (unweighted) classification. Another follow-up work to this one is that of Chen and Rudin (2018), who create a customized branch-and-bound solver for rule lists where the proportions of positive observations are constrained to be monotonically decreasing as we go down the list (called falling rule lists, also see Wang and Rudin, 2015). The work of Goh and Rudin (2014) uses mixed-integer programming to provide disjunctions of rules (rather than IF-THEN statements), and would have similar advantages to the method proposed in this work. A Bayesian counterpart to the work of Goh and Rudin (2014) is that of Wang et al (2017), which proves some results analogous to those in Section 6 for the Bayesian disjunctions of rules case.

Several recent works (Norouzi et al, 2015; Verwer and Zhang, 2017; Bertsimas and Dunn, 2017) aim at optimization of decision trees, but not for the sake of interpretability. Norouzi et al (2015) creates a smooth surrogate from a fixed tree structure to optimize for the splits, whereas Verwer and Zhang (2017) create an integer programming formulation for learning a full decision tree with a fixed depth that minimizes error rate, where each split involves one binary feature decision, and binary decision variables are used to encode every possible tree; this approach could be very computationally demanding. The method of Bertsimas and Dunn (2017) that uses integer programming to optimize trees is similarly extremely demanding, because the full set of splits need to be optimized. In contrast, we need only choose a subset of rules and their order in the rule ordering step, and we use the pre-mining of rules to reduce the size of the search space. Their method needs to be initialized with CART's solution; otherwise their algorithm could realistically perform worse than CART in practical situations, even with a few hours run time.

Learning theory bounds for classification are tightest when we optimize on the training set over a fixed class of functions. Thus, in order to achieve the tightest possible bound on out-of-sample error for functions in the class, we optimize training error. Learning theory results for rule lists can be found in the work of Rudin et al (2013); namely that the VC dimension of a set of rule list classifiers is the number of rules pre-mined.

One of the main benefits of the approach presented here is the advantage of customization. We can change the objective without having to develop a new branch and bound algorithm. Similarly, we can place additional constraints or costs in the formulation. We could place a constraint on the false positive rate. Any of these types of constraints, particularly combinations of constraints, would be extremely difficult to achieve using any non-exact method, including decision trees with splitting and pruning, or linear modeling approaches that resort to convex optimization. It would require an immense amount of tuning in order to find a feasible (let alone optimal) solution from a decision tree method when even a few of these complex constraints are present. In contrast, placing these constraints in ORL would require one or two lines of code.

Another major disadvantage of non-mathematical programming methods, as discussed earlier, is that when a model does not perform well, it is difficult to determine whether it is a fault of the algorithm or a limitation of the form of the model. Since MIP finds optimal solutions, it allows us to determine whether an interpretable-yet-accurate model exists for a given problem, or whether it could be worthwhile to resort to a more complex classifier, or different form of classifier.

Some recent work Zhang et al (2015); Lakkaraju and Rudin (2017) has used the form of rule lists to create optimal treatment regimes, which the MIP framework provided here could be adapted to handle. These treatment regimes are more than predictive models, they are policies that could be used in practice.

This paper was inspired by ideas within Allison Chang's PhD thesis (Chang, 2012), though the formulations and experiments in this work are different.

9 Conclusion

In this work we showed how a combination of theoretical insights and optimization tools can help users to design customized solutions to special cases of the well-known optimal decision tree problem; when regularization is included, the nature of the problem changes and it can be solved. Our methods can be useful for human-in-the-loop decision making problems involving predictive modeling. The models can be customized by adding constraints, so the resulting models would be directly acceptable to human experts. Without such tools, the process to create an interpretable, accurate, and constrained model can become a laborious struggle. With such tools, the end-to-end process of creating a predictive model that is acceptable can be made much easier.

Code

The code can be found at: `https://github.com/SeydaErtekin/ORL` .

Acknowledgments

We gratefully acknowledge funding from the MIT Big Data Initiative, and the National Science Foundation under grant IIS-1053407. Thanks to Daniel Bienstock and anonymous reviewers for encouragement and for helping us to improve the readability of the manuscript.

References

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Databases, pp 487–499

Angelino E, Larus-Stone N, Alabi D, Seltzer M, Rudin C (2017) Learning certifiably optimal rule lists for categorical data. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)

Angelino E, Larus-Stone N, Alabi D, Seltzer M, Rudin C (2018) Learning certifiably optimal rule lists for categorical data. Journal of Machine Learning Research 18:1–78

Anthony M (2005) Decision lists. Tech. rep., CDAM Research Report LSE-CDAM-2005-23

Bache K, Lichman M (2013) UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`

Bayardo RJ, Agrawal R (1999) Mining the most interesting rules. In: Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 145–154

Bennett KP, Blue JA (1996) Optimal decision trees. Tech. rep., R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute

Bertsimas D, Dunn J (2017) Optimal classification trees. Machine Learning (7):1039–1082

Boros E, Hammer PL, Ibaraki T, Kogan A, Mayoraz E, Muchnik I (2000) An implementation of logical analysis of data. IEEE Transactions on Knowledge and Data Engineering 12(2):292–306

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and Regression Trees. Wadsworth

Chang A (2012) Integer optimization methods for machine learning. PhD thesis, Massachusetts Institute of Technology

Chen C, Rudin C (2018) An optimization approach to learning falling rule lists. In: Proceedings of Artificial Intelligence and Statistics (AISTATS)

Chipman HA, George EI, McCulloch RE (1998) Bayesian CART model search. Journal of the American Statistical Association 93(443):935–948

Cieslak DA, Chawla NV (2008) Learning decision trees for unbalanced data. In: Daelemans W, Goethals B, Morik K (eds) Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science, vol 5211, Springer Berlin Heidelberg, pp 241–256

Cohen WW (1995) Fast effective rule induction. In: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann, pp 115–123

Cusick GR, Courtney ME, Havlicek J, Hess N (2010) Crime during the Transition to Adulthood: How Youth Fare as They Leave Out-of-Home Care. National Institute of Justice, Office of Justice Programs, US Department of Justice
Dobkin D, Fulton T, Gunopulos D, Kasif S, Salzberg S (1996) Induction of shallow decision trees
Farhangfar A, Greiner R, Zinkevich M (2008) A fast way to produce optimal fixed-depth decision trees. In: International Symposium on Artificial Intelligence and Mathematics (ISAIM 2008), Fort Lauderdale, Florida, USA, January 2-4, 2008
Fawcett T (2008) Prie: a system for generating rulelists to maximize roc performance. Data Mining and Knowledge Discovery 17(2):207–224
Freitas AA (2014) Comprehensible classification models: a position paper. ACM SIGKDD Explorations Newsletter 15(1):1–10
Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences 55(1):119–139
Friedman JH, Popescu BE (2008) Predictive learning via rule ensembles. The Annals of Applied Statistics 2(3):916–954
Geng L, Hamilton HJ (2006) Interestingness measures for data mining: a survey. ACM Computing Surveys 38
Goethals B (2003) Survey on frequent pattern mining. Tech. rep., Helsinki Institute for Information Technology
Goh ST, Rudin C (2014) Box drawings for learning with imbalanced data. In: Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)
Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: An update. SIGKDD Explor Newsl 11(1):10–18, DOI 10.1145/1656274.1656278, URL `http://doi.acm.org/10.1145/1656274.1656278`
Han J, Cheng H, Xin D, Yan X (2007) Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery 15:55–86
Hata I, Veloso A, Ziviani N (2013) Learning accurate and interpretable classifiers using optimal multi-criteria rules. Journal of Information and Data Management 4(3)
Hipp J, Güntzer U, Nakhaeizadeh G (2000) Algorithms for association rule mining: a general survey and comparison. SIGKDD Explorations 2:58–64
Huysmans J, Dejaeger K, Mues C, Vanthienen J, Baesens B (2011) An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. Decision Support Systems 51(1):141–154
Jennings DL, Amabile TM, Ross L (1982) Informal covariation assessments: Data-based versus theory-based judgements. In: Kahneman D, Slovic P, Tversky A (eds) Judgment Under Uncertainty: Heuristics and Biases, Cambridge Press, Cambridge, MA, pp 211–230
Klivans AR, Servedio RA (2006) Toward attribute efficient learning of decision lists and parities. Journal of Machine Learning Research 7:587–602
Kuhn M, Weston S, Coulter N (2012) C50: C5.0 Decision Trees and Rule-Based Models, C code for C5.0 by R. Quinlan. URL `http://CRAN.R-project.org/package=C50`, r package version 0.1.0-013
Lakkaraju H, Rudin C (2017) Learning cost effective and interpretable treatment regimes in the form of rule lists. In: Proceedings of Artificial Intelligence and Statistics (AISTATS)
Leondes CT (2002) Expert systems: the technology of knowledge management and decision making for the 21st century. Academic Press
Letham B, Rudin C, McCormick TH, Madigan D (2015) Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. Annals of Applied Statistics 9(3):1350–1371
Li W, Han J, Pei J (2001) CMAR: Accurate and efficient classification based on multiple class-association rules. IEEE International Conference on Data Mining pp 369–376
Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. In: Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, pp 80–96
Long PM, Servedio RA (2007) Attribute-efficient learning of decision lists and linear threshold functions under unconcentrated distributions. In: Advances in Neural Information Processing Systems (NIPS), vol 19, pp 921–928
Malioutov D, Varshney K (2013) Exact rule learning via boolean compressed sensing. In: Proceedings of The 30th International Conference on Machine Learning, pp 765–773
Marchand M, Sokolova M (2005) Learning with decision lists of data-dependent features. Journal of Machine Learning Research 6:427–451
McCormick TH, Rudin C, Madigan D (2012) Bayesian hierarchical modeling for predicting medical conditions. The Annals of Applied Statistics 6(2):652–668
McGarry K (2005) A survey of interestingness measures for knowledge discovery. The Knowledge Engineering Review 20:39–61
Meinshausen N (2010) Node harvest. The Annals of Applied Statistics 4(4):2049–2072
Miller GA (1956) The magical number seven, plus or minus two: Some limits to our capacity for processing information. The Psychological Review 63(2):81–97
Muggleton S, De Raedt L (1994) Inductive logic programming: Theory and methods. The Journal of Logic Programming 19:629–679
Naumov G (1991) NP-completeness of problems of construction of optimal decision trees. Soviet Physics: Doklady 36(4):270–271
Nijssen S, Fromont E (2007) Mining optimal decision trees from itemset lattices. In: Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)
Nijssen S, Fromont E (2010) Optimal constraint-based decision tree induction from itemset lattices. Data Mining and Knowledge Discovery 21(1):9–51
Norouzi M, Collins M, Johnson MA, Fleet DJ, Kohli P (2015) Efficient non-greedy optimization of decision trees. In: Advances in Neural Information Processing Systems (NIPS) 28, pp 1729–1737
Plate TA (1999) Accuracy versus interpretability in flexible modeling: Implementing a tradeoff using gaussian process models. Behaviormetrika 26:29–50
Quinlan JR (1993) C4.5: Programs for Machine Learning. Morgan Kaufmann
Ridgeway G (2013) The pitfalls of prediction. NIJ Journal, National Institute of Justice 271:34–40
Rivest RL (1987) Learning decision lists. Machine Learning 2(3):229–246
Rückert U (2008) A statistical approach to rule learning. PhD thesis, Technischen Universität München

Rudin C, Letham B, Salleb-Aouissi A, Kogan E, Madigan D (2011) Sequential event prediction with association rules. In: Proceedings of the 24th Annual Conference on Learning Theory (COLT)
Rudin C, Letham B, Madigan D (2013) Learning theory analysis for association rules and sequential event prediction. Journal of Machine Learning Research 14:3384–3436
Rüping S (2006) Learning interpretable models. PhD thesis, Universität Dortmund
Simon GJ, Kumar V, Li PW (2011) A simple statistical model and association rule filtering for classification. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 823–831
Su G, Wei D, Varshney KR, Malioutov DM (2016) Interpretable two-level boolean rule learning for classification. ArXiv e-prints 1606.05798
Tan PN, Kumar V (2000) Interestingness measures for association patterns: a perspective. Tech. rep., Department of Computer Science, University of Minnesota
Thabtah F (2007) A review of associative classification mining. The Knowledge Engineering Review 22:37–65
Ustun B, Rudin C (2016) Supersparse Linear Integer Models for Optimized Medical Scoring Systems. Machine Learning 102(3):349–391
Ustun B, Rudin C (2017) Optimized risk scores. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
Vanhoof K, Depaire B (2010) Structure of association rule classifiers: a review. In: Proceedings of the International Conference on Intelligent Systems and Knowledge Engineering (ISKE), pp 9–12
Vapnik V (1998) Statistical Learning Theory. Wiley, New York
Vellido A, Martín-Guerrero JD, Lisboa PJ (2012) Making machine learning models interpretable. In: Proc. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning
Verwer S, Zhang Y (2017) Learning decision trees with flexible constraints and objectives using integer optimization
Wang F, Rudin C (2015) Falling rule lists. In: Proceedings of Artificial Intelligence and Statistics (AISTATS)
Wang T, Rudin C, Doshi-Velez F, Liu Y, Klampfl E, MacNeille P (2017) A Bayesian framework for learning rule sets for interpretable classification. Journal of Machine Learning Research 18(70):1–37
Wu Y, Tjelmeland H, West M (2007) Bayesian CART: prior specification and posterior simulation. Journal of Computational and Graphical Statistics 16(1):44–66
Yang H, Rudin C, Seltzer M (2017) Scalable Bayesian rule lists. In: Proceedings of the 34th International Conference on Machine Learning (ICML)
Yin X, Han J (2003) CPAR: Classification based on predictive association rules. In: Proceedings of the 2003 SIAM International Conference on Data Mining, pp 331–335
Zeng J, Ustun B, Rudin C (2017) Interpretable classification models for recidivism prediction. Journal of the Royal Statistical Society: Series A (Statistics in Society) 180(3):689–722
Zhang Y, Laber EB, Tsiatis A, Davidian M (2015) Using decision lists to construct interpretable and parsimonious treatment regimes. Biometrics 71(4):895–904

A Additional Accuracy Comparison Experiments

Table 10 shows more detail about experiments, specifically, it contains the numerical values for accuracy for all algorithms, and all pairwise hypothesis tests. Because there can be a large number of parameters to tune in several of the algorithms in Table 10, it is clearly possible to tune them to provide better performance; for instance, in our method there are tuning parameters that govern the number and characteristics of rules in each class, along with tuning parameters for regularization. We chose a single parameter setting for our method for experimental comparisons to other methods, to avoid the possibility that the method performs well due to its flexibility. Further, as Table 11 shows, for SVM with gaussian kernels, there is not a single setting of SVM parameter values that is the best for all datasets. This table also shows the range of values one obtains when using SVM with various parameter settings. Note in particular that SVM never has perfect test accuracy on the Tic Tac Toe dataset, for any parameter settings we tried.

B CART and C5.0 have difficulty with the Tic Tac Toe dataset

Figures 5 and 7 show decision trees for CART and C5.0, which are not particularly interpretable. Even as we varied C5.0 and CART's parameters across their full ranges, they were not able to detect the pattern, as shown in Figures 6 and 8.

C ORL Tic Tac Toe Models for Other Folds

The ORL models for other folds are shown in Tables 12 and 13. ORL provides correct models on all folds.

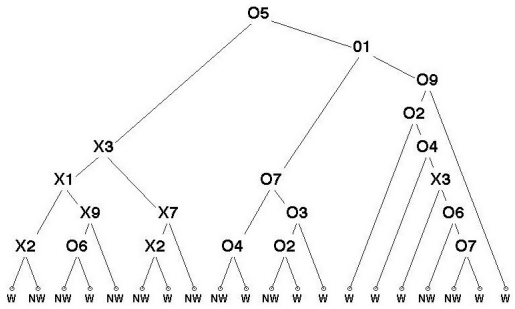


Fig. 5 CART Classifier for Tic Tac Toe dataset. Numbers indicate which square on the board is considered. Left branch always means "yes" and right branch means "no". In the leaves, "W" and "NW" stand for win or no-win for X.

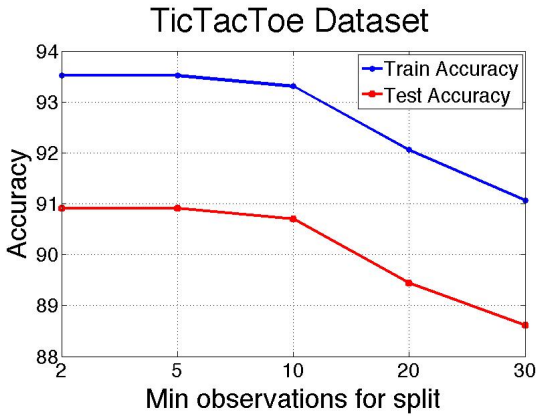


Fig. 6 CART training and testing classification accuracy on Tic Tac Toe dataset for various "confidence factor" values. Performance is not close to perfect no matter what parameter value is chosen, even though a perfect solution exists.

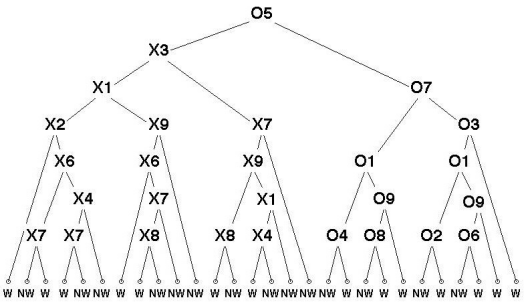


Fig. 7 C5.0 Classifier for Tic Tac Toe dataset.

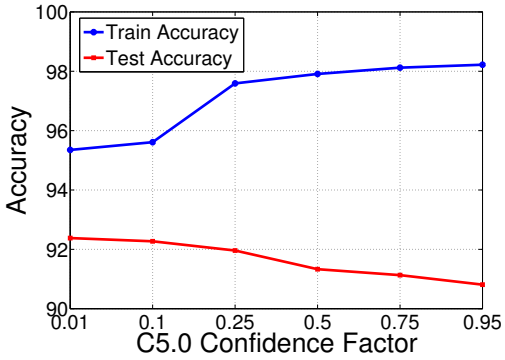


Fig. 8 C5.0 training and testing classification accuracy on Tic Tac Toe dataset for various "Confidence Factor" values. Performance is not close to perfect no matter what parameter value is chosen, even though a perfect solution exists.

		ADA	LR	SVM	RF	CART	C5.0	ORL
B.Cancer	Train	97.00	96.70	98.17	98.46	95.61	96.49	98.24
	Test	95.46	95.75	**95.90**	95.75	95.16	95.02	95.31
	ttest	h=0 p=0.5815	h=0 p=0.6686	–	h=0 p=0.8243	h=0 p=0.2999	h=0 p=0.1835	h=0 p=0.6337
CarEval	Train	99.83	95.23	98.47	99.91	96.44	99.05	98.06
	Test	**98.96**	94.73	96.76	98.32	95.78	98.32	96.53
	t-test	–	h=1 p=0.220	h=1 p=0.0048	h=0 p=0.2354	h=0 p=0.0549	h=0 p=0.2123	h=0 p=0.2254
Crx	Train	92.50	89.89	86.45	98.01	89.36	92.88	90.05
	Test	87.60	**87.75**	86.37	86.83	84.99	86.22	86.52
	t-test	h=0 p=0.8097	–	h=0 p=0.0951	h=0 p=0.0738	h=1 p=0.0350	h=1 p=0.0379	h=0 p=0.5030
Haberman	Train	77.78	77.45	79.58	81.54	76.80	76.63	80.39
	Test	73.53	73.53	72.88	73.20	71.24	71.57	**74.51**
	t-test	h=0 p=0.4774	h=0 p=0.2253	h=0 p=0.4225	h=0 p=0.0571	h=0 p=0.0635	h=0 p=0.1215	–
Housing	Train	92.29	89.53	84.78	98.22	88.14	92.19	94.76
	Test	86.36	85.37	83.79	**86.56**	83.40	84.58	85.38
	t-test	h=0 p=0.6654	h=0 p=0.0737	h=1 p=0.0052	–	h=0 p=0.1022	h=1 p=0.0380	h=0 p=0.4788
Mammo	Train	85.36	85.18	86.87	88.49	83.79	84.16	85.90
	Test	83.37	**83.86**	82.05	83.13	83.37	82.41	83.26
	t-test	h=0 p=0.2701	–	h=0 p=0.0817	h=0 p=0.5098	h=0 p=0.3825	h=0 p=0.1832	h=0 p=0.4455
Nursery	Train	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	Test	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
	t-test	–	–	–	–	–	–	–
TicTacToe	Train	99.22	98.33	99.43	100.00	92.07	97.60	100.00
	Test	97.18	98.33	96.87	97.70	89.45	91.97	**100.00**
	t-test	h=0 p=0.0823	h=0 p=0.0678	h=0 p=0.1047	h=0 p=0.1155	h=1 p=0.0406	h=0 p=0.0613	–

Table 10 Classification Accuracy (averaged over three folds). Each row represents a dataset. Bold indicates the largest value, h=0 indicates not statistically different from the best algorithm based on a matched-pairs 2 sample t-test at significance level 0.05, p-values are also reported.

D Additional Haberman experiments

In Table 14 we show the effect of varying C_1 on the accuracy of classification for one fold of the Haberman experiment, with C fixed at 1/number of rules, with a 2 hour maximum time limit for the solver (here, CPLEX). As long as C_1 is small enough the accuracy is not affected.

		$C=0.1$ $\gamma=10^{-3}$	$C=1$ $\gamma=10^{-3}$	$C=10$ $\gamma=10^{-3}$	$C=0.1$ $\gamma=10^{-2}$	$C=1$ $\gamma=10^{-2}$	$C=10$ $\gamma=10^{-2}$	$C=0.1$ $\gamma=10^{-1}$	$C=1$ $\gamma=10^{-1}$	$C=10$ $\gamma=10^{-1}$
B. Cancer	Train	84.70±1.31	95.89±1.04	96.70±0.59	96.33±0.84	97.21±0.55	98.31±0.46	93.48±0.78	98.82±0.55	98.97±0.55
	Test	85.06±0.50	95.60±2.02	95.74±1.79	95.45±2.43	95.89±2.04	96.04±1.59	93.11±2.44	95.60±1.18	95.60±1.53
CarEval	Train	70.02±0.78	71.38±1.50	90.42±4.94	70.05±0.74	93.25±0.30	95.37±0.63	73.09±1.39	99.94±0.05	100.00±0.00
	Test	70.02±1.56	71.06±3.10	90.04±4.07	70.02±1.56	92.76±0.72	93.80±0.89	70.71±2.34	97.97±0.43	99.07±0.10
Crx	Train	54.67±0.56	76.03±2.66	86.44±0.46	69.44±1.73	87.36±0.61	86.67±0.46	86.98±0.95	93.56±0.60	99.61±0.13
	Test	54.66±1.12	74.57±1.86	86.52±1.40	69.52±3.75	87.13±1.60	86.37±1.15	85.14±1.12	87.28±2.52	84.83±1.24
Haberman	Train	73.52±2.59	73.52±2.59	75.81±1.85	73.52±2.59	75.81±1.85	79.41±2.24	73.52±2.59	81.20±2.69	82.18±2.69
	Test	73.52±5.18	73.52±5.18	70.58±3.53	73.52±5.18	70.58±3.53	75.16±3.96	73.52±5.18	70.26±6.22	70.58±5.88
Housing	Train	51.97±0.89	78.85±1.34	83.99±1.86	78.26±1.75	84.38±1.92	84.98±1.92	83.30±1.93	93.87±1.24	99.30±0.16
	Test	46.63±2.85	78.05±3.96	82.99±3.44	78.05±3.36	83.78±3.84	83.98±2.60	82.20±5.20	86.55±3.62	86.75±0.95
Mammo	Train	52.95±0.84	84.57±0.64	82.89±0.26	83.85±0.82	84.03±1.92	87.10±1.71	85.12±2.82	88.73±1.06	88.97±1.10
	Test	45.54±3.97	83.13±2.30	82.53±0.52	82.04±1.43	81.56±1.29	82.77±3.10	77.71±2.67	81.08±2.06	81.20±1.89
Nursery	Train	67.83±0.30	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Test	67.83±0.61	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	99.59±0.19	100.00±0.00	100.00±0.00
TicTacToe	Train	65.34±2.59	65.34±2.59	88.62±5.80	65.34±2.59	88.25±1.70	98.32±0.39	65.34±2.59	100.00±0.00	100.00±0.00
	Test	65.35±5.17	65.35±5.17	84.42±9.66	65.35±5.17	83.50±5.51	98.32±0.79	65.35±5.17	95.72±2.07	97.07±1.72

Table 11 SVM accuracy on each dataset for various C and γ values using the RBF kernel.

1			
win	x		
60		x	
60			x

2			
win			x
58		x	
58	x		

3			
win	x	x	x
56			
56			

4			
win			
40	x	x	x
40			

5			
win		x	
48		x	
48		x	

6			
win			x
46			x
46			x

7			
win			
53			
53	x	x	x

8			
win	x		
46	x		
46	x		

9			
no win			
231			
231			

Table 12 Tic Tac Toe rules for Split 2. This model perfectly captures what it means for the X player to win in tic tac toe. If the X player gets three X's in a row in one of 8 possible ways, the classifier predicts that the X player wins. Otherwise, the X player is predicted not to win.

1			
win			x
64		x	
64	x		

2			
win			
63	x	x	x
63			

3			
win			x
55			x
55			x

4			
win	x	x	x
53			
53			

5			
win	x		
54	x		
54	x		

6			
win	x		
52		x	
52			x

7			
win			
48			
48	x	x	x

8			
win		x	
47		x	
47		x	

9			
no win			
202			
202			

Table 13 Tic Tac Toe rules for Split 3. Again this model perfectly captures what it means for the X player to win in tic tac toe.

C_1	Train accuracy	Test accuracy
0.1	79.90%	77.45%
0.2	79.90%	77.45%
0.5	79.90%	77.45%
1	75.98%	74.51%
2	75.00%	72.55%
5	74.51%	71.57%

Table 14 Train/Test accuracy for Haberman dataset experiments, with C fixed at 1/number of rules.

E Violent Crime F-scores and Gmeans

Table 15 shows numerical values for the training and test F-scores and G-means. The test values are also displayed in Figure 3.

F README for ORL Package

This package contains the data and code for running ORL experiments, associated with the paper Learning Customized and Optimized Lists of Rules with Mathematical Programming by Cynthia Rudin and Şeyda Ertekin.

In the github repository `https://github.com/SeydaErtekin/ORL`, the code for the first phase of ORL (Rule Generation) is under the Rule_Generation directory, and code for the second phase (Ranking of the discovered rules) is under the Rule_Ranking directory. We provide two of the datasets that we used in our experiments, namely Haberman's Survival and TicTacToe, under the Datasets directory.

In the package, we provide two shell scripts for running experiments with Haberman and TicTacToe datasets. The first script, run_haberman.sh, uses Haberman's sample train/test split under Datasets/processed/ and invokes the sequence of codes for generating and ranking rules, followed by displaying the ranked rules. With the default settings, the script generates the ranked rules shown in Table 5 in the paper. For TicTacToe, we use the toy ruleset under Rule_Generation/rules, so run_tictactoe.sh

		ADA	LR	SVM	RF	CART	C5.0	ORL
F-Score	Train	33.16±9.52	20.89±6.63	3.53±1.50	86.99±2.68	41.48±6.47	46.39±16.37	33.49±3.46
	Test	15.77±1.86	12.85±7.48	3.53±1.50	16.81±7.90	19.50±0.24	17.05±2.39	17.72±8.49
G-means	Train	45.04±8.25	34.50±6.37	13.22±2.78	88.50±2.60	54.77±6.88	56.53±14.32	46.94±3.44
	Test	31.08±3.06	25.96±8.65	13.22±2.78	32.72±8.83	36.59±1.88	34.58±4.88	34.45±8.92

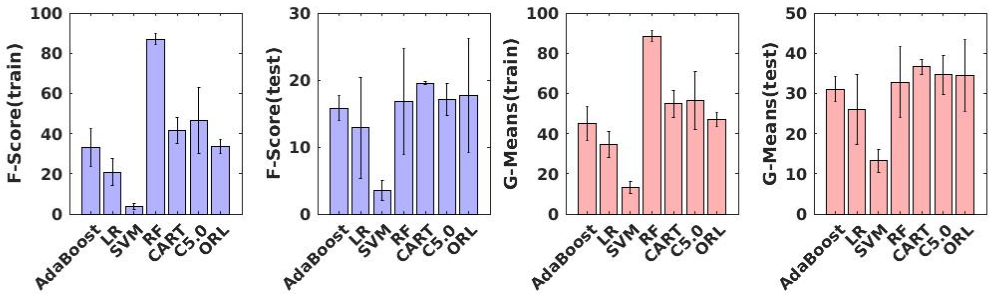
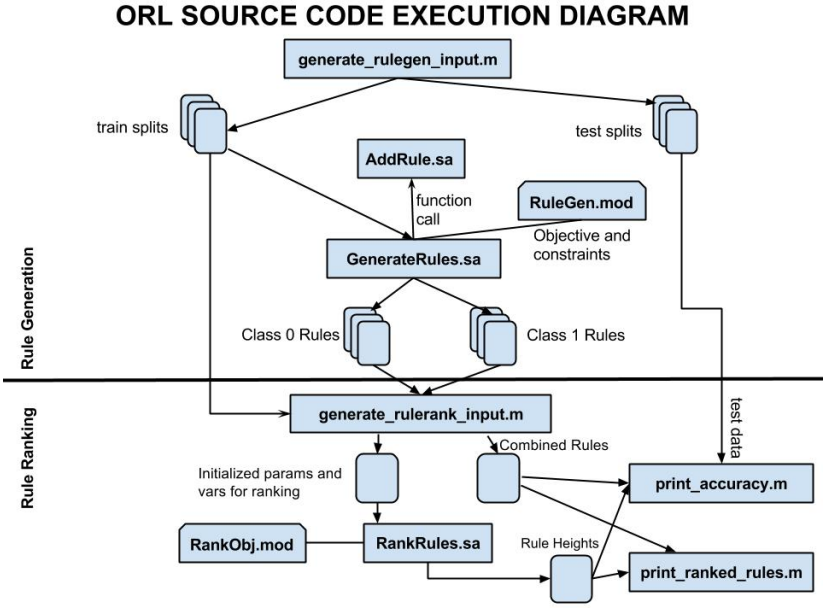


Table 15 *Top:* F-scores and G-means for the violent crime dataset (mean and standard deviation computed across folds). Each column represents an algorithm. *Bottom:* The same information (F-scores and G-means for each algorithm) is displayed.

only runs the rule ranking and displaying routines. This ruleset and corresponding results form the basis of our discussion in Section 2.1. Note that both scripts require Matlab and AMPL with Gurobi solver to be installed on the local machine.

An overview of the order of execution and the dependencies of the code is given in the diagram below.



In this package, we also provide a sample train/test split for both datasets, as well as the rules (under Rule_Generation/rules directory), the data input for rule ranking and the ranked rules (under Rule_Ranking/rules directory). The script print_ranked_rules.m can be used to view the ordered rule lists for these splits. For the Haberman's Survival dataset, the set of rules include all rules discovered with a particular setting of the input parameters. For the TicTacToe dataset, we provide the toy ruleset (that we discuss in Section 2.1 in the paper) that is a trimmed version of all discovered rules. This toy ruleset includes eight rules for the 1 class, three rules for the 0 class, and two default rules (one of each class). The input data for TicTacToe used for ranking (under Rule_Ranking/rules/tictactoe_binary_train12_rank_input.dat) only initializes the necessary parameters required for ranking; it does not need to precompute the values of the variables because the number of rules is small and the optimization completes within a few seconds.

Directory Structure

Datasets: .csv files of the original datasets. If you'd like to generate brand new train/test splits for the datasets, you can use the script generate_rulegen_input.m to generate up to 3 train/test splits by chunking the dataset into 3 equal sized chunks. Files for each split are suffixed with 12, 13, or 23, indicating which chunks were used for training. For example, the files with suffix 12 indicate that first and second chunks are in the train set and chunk 3 is in test set.

Note that due to the random shuffling of the examples, any newly generated train/test splits will be different than what we provided, hence may yield different results. If you'd like to use the existing splits that we reported results for in the paper, you can use the files under Datasets/processed.

Datasets/processed: Directory that contains train/test sets (files with .txt extension) and the train sets in ampl data format (with .dat extension). The former files are used for performance evaluation whereas the latter files are used in rule generation.

Rule_Generation: Contains generate_rulegen_input.m script for generating files under Datasets/processed, and the ampl code(s) that implement rule generation routines. GenerateRules.sa is the main implementation of the rule generation routine and AddRule.sa is a helper script (called from GenerateRules.sa) that is responsible for writing discovered rules to the output file as well as adding the rule to the list of constraints so we do not discover the same rule again in subsequent iterations. The objective and constraints for rule generation are specified in a model file called RuleGen.mod.

Rule_Generation/rules: Contains the files for the discovered rules for both classes in the datasets. We provide representative rules for both datasets in this directory. Files with "one" and "zero" suffixes include rules for one and zero classes, respectively. The file with "all" suffix is the aggregate of both files and default rules for both classes.

Rule_Ranking: Contains matlab script generate_rulerank_input.m for aggregating the rules for both classes under Rule_Generation/rules. The aggregate rules are written to Rule_Generation/rules (with "all" suffix and .txt extension) and an ampl formatted version is written under the rules subdirectory. The Rule_Ranking directory also includes the ampl code RankRules.sa that implements the rule ranking routine and the model file RankObj.mod.

Rule_Ranking/rules: Contains the data input used for rule ranking as well as the ranking output (the π vector of rule heights). This directory contains the ranked rules for both dataset at obtained for different C and C_1 settings. Running print_ranked_rules.m (up in the Rule_Ranking directory) prints the ranked rules for the specified dataset/experiment in human-readable form. print_accuracy.m similarly computes the accuracy on train or test set (controlled within the code) for the specified dataset/experiment.