Learning Certifiably Optimal Rule Lists

Elaine Angelino
EECS, UC Berkeley
Berkeley, CA 94720
elaine@eecs.berkeley.edu

Nicholas Larus-Stone
Daniel Alabi, Margo Seltzer
SEAS, Harvard University
Cambridge, MA 02138
nlarusstone@college.harvard.edu
{alabid@g, margo@eecs}.harvard.edu

Cynthia Rudin
Duke University
Durham, NC 27708
cynthia@cs.duke.edu

ABSTRACT

We present the design and implementation of a custom discrete optimization technique for building rule lists over a categorical feature space. Our algorithm provides the optimal solution, with a certificate of optimality. By leveraging algorithmic bounds, efficient data structures, and computational reuse, we achieve several orders of magnitude speedup in time and a massive reduction of memory consumption. We demonstrate that our approach produces optimal rule lists on practical problems in seconds. This framework is a novel alternative to CART and other decision tree methods.

CCS CONCEPTS

• **Computing methodologies → Discrete space search**; **Classification and regression trees**;

KEYWORDS

Rule lists; Decision trees; Optimization; Interpretable models

1 INTRODUCTION

As machine learning continues to gain prominence in socially-important decision-making, the interpretability of predictive models remains a crucial problem. Our goal is to build models that are both highly predictive and easily understood by humans. We use rule lists, also known as decision lists, to achieve this goal. Rule lists are lists composed of if-then statements, which are easily interpreted; the rules give a reason for each prediction (Figure 1).

Constructing rule lists, or more generally, decision trees, has been a challenge for more than 30 years; most approaches use greedy splitting techniques [4, 32, 34]. Recent approaches use Bayesian analysis, either to find a locally optimal solution [6] or to explore the search space [25, 45]. These approaches achieve high accuracy while also managing to run reasonably quickly. However, despite the apparent accuracy of the rule lists generated by these algorithms, there is no way to determine either if the generated rule list is optimal or how close it is to optimal, where optimality is defined with respect to minimization of a regularized loss function.

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098047

Optimality is important, because there are societal implications for a lack of optimality. Consider the recent ProPublica article on the COMPAS recidivism prediction tool [23]. It highlights a case where a black-box, proprietary predictive model is being used for recidivism prediction. The authors show that the COMPAS scores are racially biased, but since the model is not transparent, no one (outside of the creators of COMPAS) can determine the reason or extent of the bias [23], nor can anyone determine the reason for any particular prediction. By using COMPAS, users implicitly assumed that a transparent model would not be sufficiently accurate for recidivism prediction, *i.e.*, they assumed that a black box model would provide better accuracy. We wondered whether there was indeed no transparent and sufficiently accurate model. Answering this question requires solving a computationally hard problem. Namely, we would like to both find a transparent model that is optimal within a particular pre-determined class of models and produce a certificate of its optimality, with respect to the regularized empirical risk. This would enable one to say, for this problem and model class, with certainty and before resorting to black box methods, whether there exists a transparent model. While there may be differences between training and test performance, finding the simplest model with optimal training performance is prescribed by statistical learning theory.

To that end, we consider the class of rule lists assembled from pre-mined frequent itemsets and search for an optimal rule list that minimizes a regularized risk function, R. This is a hard discrete optimization problem. Brute force solutions that minimize R are computationally prohibitive due to the exponential number of possible rule lists. However, this is a worst case bound that is not realized in practical settings. For realistic cases, it is possible to solve fairly large cases of this problem to optimality, with the careful use of algorithms, data structures, and implementation techniques.

We develop specialized tools from the fields of discrete optimization and artificial intelligence. Specifically, we introduce a special branch-and-bound algorithm, called Certifiably Optimal RulE ListS (CORELS), that provides (1) the optimal solution, (2) a certificate of optimality, and (3) optionally, a collection of near-optimal solutions and the distance between each such solution and the optimal one. The certificate of optimality means that we can investigate how

if ($age = 23 - 25$) **and** ($priors = 2 - 3$) **then predict** *yes*
else if ($age = 18 - 20$) **then predict** *yes*
else if ($sex = male$) **and** ($age = 21 - 22$) **then predict** *yes*
else if ($priors > 3$) **then predict** *yes*
else predict *no*

Figure 1: An example rule list that predicts two-year recidivism for the ProPublica dataset, found by CORELS.

close other models (*e.g.*, models provided by greedy algorithms) are to optimal. In particular, we can investigate if the rule lists from probabilistic approaches are nearly optimal or whether those approaches sacrifice too much accuracy in the interest of speed.

The efficacy of CORELS depends on how much of the search space our bounds allow us to prune; we seek a tight lower bound on R. The bound we maintain throughout execution is a maximum of several bounds that come in three categories. The first category of bounds are those intrinsic to the rules themselves. This category includes bounds stating that each rule must capture sufficient data; if not, the rule list is provably non-optimal. The second type of bound compares a lower bound on the value of R to that of the current best solution. This allows us to exclude parts of the search space that could never be better than our current solution. Finally, our last type of bound is based on comparing incomplete rule lists that capture the same data and allows us to pursue only the most accurate option. This last class of bounds is especially important – without our use of a novel *symmetry-aware map*, we are unable to solve most problems of reasonable scale. This symmetry-aware map keeps track of the best accuracy over all observed permutations of a given incomplete rule list.

We keep track of these bounds using a modified *prefix tree*, a data structure also known as a trie. Each node in the prefix tree represents an individual rule; thus, each path in the tree represents a rule list such that the final node in the path contains metrics about that rule list. This tree structure, together with a search policy and sometimes a queue, enables a variety of strategies, including breadth-first, best-first, and stochastic search. In particular, we can design different best-first strategies by customizing how we order elements in a priority queue. In addition, we are able to limit the number of nodes in the tree and thereby enable tuning of space-time tradeoffs in a robust manner. This tree structure is a useful way of organizing the generation and evaluation of rule lists.

CORELS targets large (not massive) problems, where interpretability and certifiable optimality are important. We illustrate the efficacy of our approach using (1) the ProPublica COMPAS dataset [23], for the problem of two-year recidivism prediction, and (2) the New York Civil Liberties Union (NYCLU) 2014 stop-and-frisk dataset [30], to predict whether a weapon will be found on a stopped individual who is frisked or searched. We produce certifiably optimal, interpretable rule lists that achieve the same accuracy as approaches such as random forests. This calls into question the need for use of a proprietary, black box algorithm for recidivism prediction.

Our work overlaps with the thesis presented by Larus-Stone [24]. We have also written a long version of this report that includes proofs to all bounds in §3, additional bounds and empirical results, and further implementation and data processing details [1].

Our code is at **https://github.com/nlarusstone/corels**.

2 RELATED WORK

Since every rule list is a decision tree and every decision tree can be expressed as an equivalent rule list, the problem we are solving is a version of the "optimal decision tree" problem, though regularization changes the nature of the problem (as shown through our bounds). The optimal decision tree problem is computationally hard, though since the late 1990's, there has been research on building optimal decision trees using optimization techniques [2, 13, 14]. A particularly interesting paper along these lines is that of Nijssen and Fromont [31], who created a "bottom-up" way to form optimal decision trees. Their method performs an expensive search step, mining all possible leaves (rather than all possible rules), and uses those leaves to form trees. Their method can lead to memory problems, but it is possible that these memory issues can be mitigated using the theorems in this paper.[1] None of these methods used the tight bounds and data structures of CORELS.

Because the optimal decision tree problem is hard, there are a huge number of algorithms such as CART [4] and C4.5 [32] that do not perform exploration of the search space beyond greedy splitting. Similarly, there are decision list and associative classification methods that construct rule lists iteratively in a greedy way [26–28, 34, 37, 40, 41, 46]. Some exploration of the search space is done by Bayesian decision tree methods [7, 8, 12] and Bayesian rule-based methods [25, 45]. The space of trees of a given depth is much larger than the space of rule lists of that same depth, and the trees within the Bayesian tree algorithms are grown in a top-down greedy way. Because of this, authors of Bayesian tree algorithms have noted that their MCMC chains tend to reach only locally optimal solutions. The RIPPER algorithm [10] is similar to the Bayesian tree methods in that it grows, prunes, and then locally optimizes. The space of rule lists is smaller than that of trees, and has simpler structure. Consequently, Bayesian rule list algorithms tend to be more successful at escaping local minima and can introduce methods of exploring the search space that exploit this structure—these properties motivate our focus on lists. That said, the tightest bounds for the Bayesian lists (namely those of Yang et al. [45], upon whose work we build), are not nearly as tight as those of CORELS.

Tight bounds, on the other hand, have been developed for the (immense) literature on building disjunctive normal form (DNF) models, a good example of this is the work of Rijnbeek and Kors [33]. For models of a given size, since the class of DNF's is a proper subset of decision lists, our framework can be restricted to learn optimal DNF's. The field of DNF learning includes work from the fields of rule learning/induction (e.g., early algorithms [9, 15, 29]) and associative classification [41]. Most papers in these fields aim to carefully guide the search through the space of models. If we were to place a restriction on our code to learn DNF's, which would require restricting predictions within the list to the positive class only, we could potentially use methods from rule learning and associative classification to help order CORELS' queue, which would in turn help us eliminate parts of the search space more quickly.

Some of our bounds, including the minimum support bound (§3.7, Theorem 3.8), come from Rudin and Ertekin [36], who provide flexible mixed-integer programming (MIP) formulations using the same objective as we use here; MIP solvers in general cannot compete with the speed of CORELS.

CORELS depends on pre-mined rules, which we obtain here via enumeration. The literature on association rule mining is huge, and any method for rule mining could be reasonably substituted.

CORELS' main use is for producing interpretable predictive models. There is a growing interest in interpretable (transparent,

[1]There is no public version of their code for distribution as of this writing.

comprehensible) models because of their societal importance (see [3, 11, 16, 18, 20, 21, 38, 39, 42]). There are now regulations on algorithmic decision-making in the European Union on the "right to an explanation" [19] that would legally require interpretability of predictions. There is work in both the DNF literature [35] and decision tree literature [17] on building interpretable models. Interpretable models must be so sparse that they need to be heavily optimized; heuristics tend to produce either inaccurate or non-sparse models.

Interpretability has many meanings, and it is possible to extend the ideas in this work to other definitions of interpretability; these rule lists may have exotic constraints that help with ease-of-use. For example, Falling Rule Lists [44] are constrained to have decreasing probabilities down the list, and thus could be easier to use. We are currently working on bounds for Falling Rule Lists [5] similar to those presented here, but even CORELS' basic support bounds do not hold for the falling case.

The models produced by CORELS are predictive only; they cannot be used for policy-making. It is possible to adapt CORELS' framework for causal inference [43], dynamic treatment regimes [47], or cost-sensitive dynamic treatment regimes [22] to help with policy design. Both Wang and Rudin [44] and Lakkaraju and Rudin [22] use Monte Carlo searches to explore the space of rule lists. CORELS could potentially be adapted to handle these kinds of interesting problems.

3 LEARNING OPTIMAL RULE LISTS

3.1 Notation

We restrict our setting to binary classification. Let $\{(x_n, y_n)\}_{n=1}^N$ denote training data, where $x_n \in \{0,1\}^J$ are binary features and $y_n \in \{0,1\}$ are labels. Let $\mathbf{x} = \{x_n\}_{n=1}^N$ and $\mathbf{y} = \{y_n\}_{n=1}^N$, and let $x_{n,j}$ denote the j-th feature of x_n.

A rule list $d = (r_1, r_2, \dots, r_K, r_0)$ of length $K \geq 0$ is a $(K+1)$-tuple consisting of K distinct association rules, $r_k = p_k \to q_k$, for $k = 1, \dots, K$, followed by a default rule r_0. Figure 1 illustrates a 4-rule list, $d = (r_1, r_2, r_3, r_4, r_0)$. An association rule $r = p \to q$ is an implication corresponding to the conditional statement, "if p, then q." In our setting, an antecedent p is a Boolean assertion that evaluates to either true or false for each datum x_n, and a consequent q is a label prediction. For example, $(x_{n,1} = 0) \wedge (x_{n,3} = 1) \to (y_n = 1)$ is an association rule. The final default rule r_0 in a rule list can be thought of as a association rule $p_0 \to q_0$ whose antecedent p_0 simply asserts true.

Let $d = (r_1, r_2, \dots, r_K, r_0)$ be a rule list, where $r_k = p_k \to q_k$ for each $k = 0, \dots, K$. We introduce a useful alternate rule list representation: $d = (d_p, \delta_p, q_0, K)$, where we define $d_p = (p_1, \dots, p_K)$ to be d's prefix, $\delta_p = (q_1, \dots, q_K) \in \{0,1\}^K$ gives the label predictions associated with d_p, and $q_0 \in \{0,1\}$ is the default label prediction. For the rule list in Figure 1, we would write $d = (d_p, \delta_p, q_0, K)$, where $d_p = (p_1, p_2, p_3, p_4)$, $\delta_p = (1,1,1,1)$, $q_0 = 0$, and $K = 4$.

Let $d_p = (p_1, \dots, p_k, \dots, p_K)$ be an antecedent list, then for any $k \leq K$, we define $d_p^k = (p_1, \dots, p_k)$ to be the k-prefix of d_p. For any such k-prefix d_p^k, we say that d_p starts with d_p^k. For any given space of rule lists, we define $\sigma(d_p)$ to be the set of all rule lists whose prefixes start with d_p:

$$\sigma(d_p) = \{(d'_p, \delta'_p, q'_0, K') : d'_p \text{ starts with } d_p\}. \quad (1)$$

If $d_p = (p_1, \dots, p_K)$ and $d'_p = (p_1, \dots, p_K, p_{K+1})$ are two prefixes such that d'_p starts with d_p and extends it by a single antecedent, we say that d_p is the parent of d'_p and that d'_p is a child of d_p.

A rule list d classifies datum x_n by providing the label prediction q_k of the first rule r_k whose antecedent p_k is true for x_n. We say that an antecedent p_k of antecedent list d_p captures x_n in the context of d_p if p_k is the first antecedent in d_p that evaluates to true for x_n. A prefix captures those data captured by its antecedents; for a rule list $d = (d_p, \delta_p, q_0, K)$, data not captured by the prefix d_p are classified according to the default label prediction q_0.

Let β be a set of antecedents. We define $\text{cap}(x_n, \beta) = 1$ if an antecedent in β captures datum x_n, and 0 otherwise. For example, let d_p and d'_p be prefixes such that d'_p starts with d_p, then d'_p captures all the data that d_p captures: $\{x_n : \text{cap}(x_n, d_p)\} \subseteq \{x_n : \text{cap}(x_n, d'_p)\}$.

Now let d_p be an ordered list of antecedents, and let β be a subset of antecedents in d_p. Let us define $\text{cap}(x_n, \beta \,|\, d_p) = 1$ if β captures datum x_n in the context of d_p, *i.e.*, if the first antecedent in d_p that evaluates to true for x_n is an antecedent in β, and 0 otherwise. Thus, $\text{cap}(x_n, \beta \,|\, d_p) = 1$ only if $\text{cap}(x_n, \beta) = 1$; $\text{cap}(x_n, \beta \,|\, d_p) = 0$ either if $\text{cap}(x_n, \beta) = 0$, or if $\text{cap}(x_n, \beta) = 1$ but there is an antecedent α in d_p, preceding all antecedents in β, such that $\text{cap}(x_n, \alpha) = 1$. For example, if $d_p = (p_1, \dots, p_k, \dots, p_K)$ is a prefix, then

$$\text{cap}(x_n, p_k \,|\, d_p) = \left(\bigwedge_{k'=1}^{k-1} \neg\, \text{cap}(x_n, p_{k'}) \right) \wedge \text{cap}(x_n, p_k)$$

indicates whether antecedent p_k captures datum x_n in the context of d_p. Now, define $\text{supp}(\beta, \mathbf{x})$ to be the normalized support of β,

$$\text{supp}(\beta, \mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, \beta), \quad (2)$$

and similarly define $\text{supp}(\beta, \mathbf{x} \,|\, d_p)$ to be the normalized support of β in the context of d_p,

$$\text{supp}(\beta, \mathbf{x} \,|\, d_p) = \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, \beta \,|\, d_p), \quad (3)$$

Next, we address how empirical data constrains rule lists. Given training data $(\mathbf{x}, \mathbf{y})$, an antecedent list $d_p = (p_1, \dots, p_K)$ implies a rule list $d = (d_p, \delta_p, q_0, K)$ with prefix d_p, where the label predictions $\delta_p = (q_1, \dots, q_K)$ and q_0 are empirically set to minimize the number of misclassification errors made by the rule list on the training data. Thus for $1 \leq k \leq K$, label prediction q_k corresponds to the majority label of data captured by antecedent p_k in the context of d_p, and the default q_0 corresponds to the majority label of data not captured by d_p. In the remainder of our presentation, whenever we refer to a rule list with a particular prefix, we implicitly assume these empirically determined label predictions.

Our method is technically an associative classification method since it leverages pre-mined rules.

3.2 Objective function

Define a simple objective function for a rule list $d = (d_p, \delta_p, q_0, K)$:

$$R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda K. \quad (4)$$

This objective function is a regularized empirical risk; it consists of a loss $\ell(d, \mathbf{x}, \mathbf{y})$, measuring misclassification error, and a regularization term that penalizes longer rule lists. $\ell(d, \mathbf{x}, \mathbf{y})$ is the fraction of training data whose labels are incorrectly predicted by d. In our setting, the regularization parameter $\lambda \geq 0$ is a small constant; *e.g.*, $\lambda = 0.01$ can be thought of as adding a penalty equivalent to misclassifying 1% of data when increasing a rule list's length by one.

3.3 Optimization framework

Our objective has structure amenable to global optimization via a branch-and-bound framework. In particular, we make a series of important observations, each of which translates into a useful bound, and that together interact to eliminate large parts of the search space. We discuss these in depth in what follows:

- Lower bounds on a prefix also hold for every extension of that prefix. (§3.4, Theorem 3.1)
- If a rule list is not accurate enough with respect to its length, we can prune all extensions of it. (§3.4, Lemma 3.2)
- We can calculate *a priori* an upper bound on the maximum length of an optimal rule list. (§3.5, Theorem 3.5)
- Each rule in an optimal rule list must have support that is sufficiently large. This allows us to construct rule lists from frequent itemsets, while preserving the guarantee that we can find a globally optimal rule list from pre-mined rules. (§3.7, Theorem 3.8)
- Each rule in an optimal rule list must predict accurately. In particular, the number of observations predicted correctly by each rule in an optimal rule list must be above a threshold. (§3.7, Theorem 3.9)
- We need only consider the optimal permutation of antecedents in a prefix; we can omit all other permutations. (§3.8, Theorem 3.10 and Corollary 3.11)
- If multiple observations have identical features and opposite labels, we know that any model will make mistakes. In particular, the number of mistakes on these observations will be at least the number of observations with the minority label. (§3.10, Theorem 3.14)

We present additional theorems and all proofs in [1].

3.4 Hierarchical objective lower bound

We can decompose the misclassification error into two contributions corresponding to the prefix and the default rule:

$$\ell(d, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}),$$

where $d_p = (p_1, \dots, p_K)$ and $\delta_p = (q_1, \dots, q_K)$;

$$\ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \text{cap}(x_n, p_k \mid d_p) \wedge \mathbb{1}[q_k \neq y_n]$$

is the fraction of data captured and misclassified by the prefix, and

$$\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \neg\, \text{cap}(x_n, d_p) \wedge \mathbb{1}[q_0 \neq y_n]$$

is the fraction of data not captured by the prefix and misclassified by the default rule. Eliminating the latter error term gives a lower bound $b(d_p, \mathbf{x}, \mathbf{y})$ on the objective,

$$b(d_p, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K \leq R(d, \mathbf{x}, \mathbf{y}), \qquad (5)$$

where we have suppressed the lower bound's dependence on label predictions δ_p because they are fully determined, given $(d_p, \mathbf{x}, \mathbf{y})$.

Algorithm 1 Branch-and-bound for learning rule lists.

Input: Objective function $R(d, \mathbf{x}, \mathbf{y})$, objective lower bound $b(d_p, \mathbf{x}, \mathbf{y})$, set of antecedents $S = \{s_m\}_{m=1}^{M}$, training data $(\mathbf{x}, \mathbf{y}) = \{(x_n, y_n)\}_{n=1}^{N}$, initial best known rule list d^0 with objective $R^0 = R(d^0, \mathbf{x}, \mathbf{y})$

Output: Provably optimal rule list d^* with minimum objective R^*

```
$(d^c, R^c) \leftarrow (d^0, R^0)$                ▷ Initialize best rule list and objective
$Q \leftarrow$ queue( [ ( ) ] )                  ▷ Initialize queue with empty prefix
while $Q$ not empty do                           ▷ Stop when queue is empty
    $d_p \leftarrow Q$.pop( )                    ▷ Remove prefix $d_p$ from the queue
    if $b(d_p, \mathbf{x}, \mathbf{y}) < R^c$ then   ▷ Bound: Apply Theorem 3.1
        $R \leftarrow R(d, \mathbf{x}, \mathbf{y})$   ▷ Compute objective of $d_p$'s rule list $d$
        if $R < R^c$ then                        ▷ Update best rule list and objective
            $(d^c, R^c) \leftarrow (d, R)$
        end if
        for $s$ in $S$ do                        ▷ Branch: Enqueue $d_p$'s children
            if $s$ not in $d_p$ then
                $Q$.push( $(d_p, s)$ )
            end if
        end for
    end if
end while
$(d^*, R^*) \leftarrow (d^c, R^c)$                ▷ Identify provably optimal solution
```

Furthermore, $b(d_p, \mathbf{x}, \mathbf{y})$ gives a lower bound on the objective of *any* rule list whose prefix starts with d_p.

Theorem 3.1 (Hierarchical objective lower bound). *Define $b(d_p, \mathbf{x}, \mathbf{y}) = \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K$, as in (5). Also, define $\sigma(d_p)$ to be the set of all rule lists whose prefixes starts with d_p, as in (1). Let $d = (d_p, \delta_p, q_0, K)$ be a rule list with prefix d_p, and let $d' = (d'_p, \delta'_p, q'_0, K') \in \sigma(d_p)$ be any rule list such that its prefix d'_p starts with d_p and $K' \geq K$, then $b(d_p, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y})$.*

To generalize, consider a sequence of prefixes such that each prefix starts with all previous prefixes in the sequence. It follows that the corresponding sequence of objective lower bounds increases monotonically. This is precisely the structure required and exploited by branch-and-bound, illustrated in Algorithm 1.

Specifically, the objective lower bound in Theorem 3.1 enables us to prune the state space hierarchically. While executing branch-and-bound, we keep track of the current best (smallest) objective R^c, thus it is a dynamic, monotonically decreasing quantity. If we encounter a prefix d_p with lower bound $b(d_p, \mathbf{x}, \mathbf{y}) \geq R^c$, then by Theorem 3.1, we do not need to consider *any* rule list $d' \in \sigma(d_p)$ whose prefix d'_p starts with d_p. For the objective of such a rule list, the current best objective provides a lower bound, *i.e.*, $R(d', \mathbf{x}, \mathbf{y}) \geq b(d'_p, \mathbf{x}, \mathbf{y}) \geq b(d_p, \mathbf{x}, \mathbf{y}) \geq R^c$, and thus d' cannot be optimal.

Next, we state an immediate consequence of Theorem 3.1.

Lemma 3.2 (Objective lower bound with one-step lookahead). *Let d_p be a K-prefix and let R^c be the current best objective. If $b(d_p, \mathbf{x}, \mathbf{y}) + \lambda \geq R^c$, then for any K'-rule list $d' \in \sigma(d_p)$ whose prefix d'_p starts with d_p and $K' > K$, it follows that $R(d', \mathbf{x}, \mathbf{y}) \geq R^c$.*

Therefore, even if we encounter a prefix d_p with lower bound $b(d_p, \mathbf{x}, \mathbf{y}) \leq R^c$, if $b(d_p, \mathbf{x}, \mathbf{y}) + \lambda \geq R^c$, then we can prune all prefixes d'_p that start with and are longer than d_p.

3.5 Upper bounds on prefix length

At any point during branch-and-bound execution, the current best objective R^c implies an upper bound on the maximum prefix length we might still have to consider.

Theorem 3.3 (Upper bound on prefix length). *Consider a state space of all rule lists formed from a set of M antecedents. Let $L(d)$ be the length of rule list d and let R^c be the current best objective. For all optimal rule lists $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$*

$$L(d^*) \leq \min\left(\lfloor R^c/\lambda \rfloor, M\right), \tag{6}$$

where λ is the regularization parameter.

Corollary 3.4 (Simple upper bound on prefix length). *For all optimal rule lists $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$,*

$$L(d^*) \leq \min\left(\lfloor 1/2\lambda \rfloor, M\right). \tag{7}$$

For any particular prefix d_p, we can obtain potentially tighter upper bounds on prefix length for all prefixes that start with d_p.

Theorem 3.5 (Prefix-specific upper bound on prefix length). *Let $d = (d_p, \delta_p, q_0, K)$ be a rule list, let $d' = (d'_p, \delta'_p, q'_0, K') \in \sigma(d_p)$ be any rule list such that d'_p starts with d_p, and let R^c be the current best objective. If d'_p has lower bound $b(d'_p, \mathbf{x}, \mathbf{y}) < R^c$, then*

$$K' < \min\left(K + \left\lfloor \frac{R^c - b(d_p, \mathbf{x}, \mathbf{y})}{\lambda} \right\rfloor, M\right). \tag{8}$$

We can view Theorem 3.5 as a generalization of our one-step lookahead bound (Lemma 3.2), as (8) is equivalently a bound on $K' - K$, an upper bound on the number of remaining 'steps' corresponding to an iterative sequence of single-rule extensions of a prefix d_p.

3.6 Upper bounds on prefix evaluations

In this section, we use Theorem 3.5's upper bound on prefix length to derive a corresponding upper bound on the number of prefix evaluations made by Algorithm 1. We present Theorem 3.6, in which we use information about the state of Algorithm 1's execution to calculate, for any given execution state, upper bounds on the number of additional prefix evaluations that might be required for the execution to complete. The relevant execution state depends on the current best objective R^c and information about prefixes we are planning to evaluate, *i.e.*, prefixes in the queue Q of Algorithm 1. We define the number of *remaining prefix evaluations* as the number of prefixes that are currently in or will be inserted into the queue.

Theorem 3.6 (Upper bound on the number of remaining prefix evaluations). *Consider the state space of all rule lists formed from a set of M antecedents, and consider Algorithm 1 at a particular instant during execution. Let R^c be the current best objective, let Q be the queue, and let $L(d_p)$ be the length of prefix d_p. Define $\Gamma(R^c, Q)$ to be the number of remaining prefix evaluations, then*

$$\Gamma(R^c, Q) \leq \sum_{d_p \in Q} \sum_{k=0}^{f(d_p)} \frac{(M - L(d_p))!}{(M - L(d_p) - k)!}, \tag{9}$$

$$\textit{where} \quad f(d_p) = \min\left(\left\lfloor \frac{R^c - b(d_p, \mathbf{x}, \mathbf{y})}{\lambda} \right\rfloor, M - L(d_p)\right). \tag{10}$$

Proof. The number of remaining prefix evaluations is equal to the number of prefixes that are currently in or will be inserted into queue Q. For any such prefix d_p, Theorem 3.5 gives an upper bound on the length of any prefix d'_p that starts with d_p:

$$L(d'_p) \leq \min\left(L(d_p) + \left\lfloor \frac{R^c - b(d_p, \mathbf{x}, \mathbf{y})}{\lambda} \right\rfloor, M\right) \equiv U(d_p). \tag{11}$$

This gives an upper bound on the number of remaining prefix evaluations: $\Gamma(R^c, Q) \leq \sum_{d_p \in Q} \sum_{k=0}^{U(d_p) - L(d_p)} P(M - L(d_p), k)$. □

The proposition below is a naïve upper bound on the total number of prefix evaluations over the course of Algorithm 1's execution. It only depends on the number of rules and the regularization parameter λ; *i.e.*, unlike Theorem 3.6, it does not use algorithm execution state to bound the size of the search space.

Proposition 3.7 (Upper bound on the total number of prefix evaluations). *Define $\Gamma_{\text{tot}}(S)$ to be the total number of prefixes evaluated by Algorithm 1, given the state space of all rule lists formed from a set S of M rules. For any set S of M rules,*

$$\Gamma_{\text{tot}}(S) \leq \sum_{k=0}^{K} \frac{M!}{(M-k)!}, \quad \textit{where} \quad K = \min(\lfloor 1/2\lambda \rfloor, M). \tag{12}$$

Proof. By Corollary 3.4, $K \equiv \min(\lfloor 1/2\lambda \rfloor, M)$ gives an upper bound on the length of any optimal rule list. We obtain (12) by viewing our problem as finding the optimal selection and permutation of k out of M rules, over all $k \leq K$. □

3.7 Lower bounds on antecedent support

In this section, we give two lower bounds on the normalized support of each antecedent in any optimal rule list; both are related to the regularization parameter λ.

Theorem 3.8 (Lower bound on antecedent support). *Let $d^* = (d_p, \delta_p, q_0, K) \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$ be any optimal rule list, with objective R^*. For each antecedent p_k in prefix $d_p = (p_1, \ldots, p_K)$, the regularization parameter provides a lower bound, $\lambda \leq \operatorname{supp}(p_k, \mathbf{x} \mid d_p)$, on the normalized support of p_k.*

Thus, we can prune a prefix d_p if any of its antecedents captures less than a fraction λ of data, even if $b(d_p, \mathbf{x}, \mathbf{y}) < R^*$. The bound in Theorem 3.8 depends on the antecedents, but not the label predictions, and thus does not account for misclassification error. Theorem 3.9 gives a tighter bound by leveraging this information.

Theorem 3.9 (Lower bound on accurate antecedent support). *Let $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$ be any optimal rule list, with objective R^*; let $d^* = (d_p, \delta_p, q_0, K)$, with prefix $d_p = (p_1, \ldots, p_K)$ and labels $\delta_p = (q_1, \ldots, q_K)$. For each rule $p_k \rightarrow q_k$ in d^*, define a_k to be the fraction of data that are captured by p_k and correctly classified:*

$$a_k \equiv \frac{1}{N} \sum_{n=1}^{N} \operatorname{cap}(x_n, p_k \mid d_p) \wedge \mathbb{1}[q_k = y_n]. \tag{13}$$

The regularization parameter provides a lower bound, $\lambda \leq a_k$.

Thus, we can prune a prefix if any of its rules correctly classifies less than a fraction λ of data. While the lower bound in Theorem 3.8

is a sub-condition of the lower bound in Theorem 3.9, we can still leverage both – since the sub-condition is easier to check, checking it first can accelerate pruning. In addition to applying Theorem 3.8 in the context of constructing rule lists, we can furthermore apply it in the context of rule mining (§3.1). Specifically, it implies that we should only mine rules with normalized support of at least λ; we need not mine rules with a smaller fraction of observations.[2] In contrast, we can only apply Theorem 3.9 in the context of constructing rule lists; it depends on the misclassification error associated with each rule in a rule list, thus it provides a lower bound on the number of observations that each such rule must correctly classify.

3.8 Equivalent support bound

If two prefixes capture the same data, and one is more accurate than the other, then there is no benefit to considering prefixes that start with the less accurate one. Let d_p be a prefix, and consider the best possible rule list whose prefix starts with d_p. If we take its antecedents in d_p and replace them with another prefix with the same support (that could include different antecedents), then its objective can only become worse or remain the same.

Formally, let D_p be a prefix, and let $\xi(D_p)$ be the set of all prefixes that capture exactly the same data as D_p. Now, let d be a rule list with prefix d_p in $\xi(D_p)$, such that d has the minimum objective over all rule lists with prefixes in $\xi(D_p)$. Finally, let d' be a rule list whose prefix d'_p starts with d_p, such that d' has the minimum objective over all rule lists whose prefixes start with d_p. Theorem 3.10 below implies that d' also has the minimum objective over all rule lists whose prefixes start with *any* prefix in $\xi(D_p)$.

Theorem 3.10 (Equivalent support bound). *Define $\sigma(d_p)$ to be the set of all rule lists whose prefixes start with d_p, as in* (1). *Let $d = (d_p, \delta_p, q_0, K)$ be a rule list with prefix $d_p = (p_1, \dots, p_K)$, and let $D = (D_p, \Delta_p, Q_0, \kappa)$ be a rule list with prefix $D_p = (P_1, \dots, P_\kappa)$, such that d_p and D_p capture the same data, i.e.,*

$$\{x_n : \text{cap}(x_n, d_p)\} = \{x_n : \text{cap}(x_n, D_p)\}. \tag{14}$$

If the objective lower bounds of d and D obey $b(d_p, \mathbf{x}, \mathbf{y}) \le b(D_p, \mathbf{x}, \mathbf{y})$, then the objective of the optimal rule list in $\sigma(d_p)$ gives a lower bound on the objective of the optimal rule list in $\sigma(D_p)$:

$$\min_{d' \in \sigma(d_p)} R(d', \mathbf{x}, \mathbf{y}) \le \min_{D' \in \sigma(D_p)} R(D', \mathbf{x}, \mathbf{y}). \tag{15}$$

Thus, if prefixes d_p and D_p capture the same data, and their objective lower bounds obey $b(d_p, \mathbf{x}, \mathbf{y}) \le b(D_p, \mathbf{x}, \mathbf{y})$, Theorem 3.10 implies that we can prune D_p. Next, in Sections 3.9 and 3.9.1, we highlight and analyze the special case of prefixes that capture the same data because they contain the same antecedents.

3.9 Permutation bound

If two prefixes are composed of the same antecedents, *i.e.*, they contain the same antecedents up to a permutation, then they capture the same data, and thus Theorem 3.10 applies. Therefore, if one is more accurate than the other, then there is no benefit to considering prefixes that start with the less accurate one. Let d_p be a prefix, and consider the best possible rule list whose prefix starts with d_p. If we permute its antecedents in d_p, then its objective can only become worse or remain the same.

[2]We describe our application of this idea in the appendix of our long report [1].

Formally, let $P = \{p_k\}_{k=1}^K$ be a set of K antecedents, and let Π be the set of all K-prefixes corresponding to permutations of antecedents in P. Let prefix d_p in Π have the minimum prefix misclassification error over all prefixes in Π. Also, let d' be a rule list whose prefix d'_p starts with d_p, such that d' has the minimum objective over all rule lists whose prefixes start with d_p. Corollary 3.11 below, which can be viewed as special case of Theorem 3.10, implies that d' also has the minimum objective over all rule lists whose prefixes start with *any* prefix in Π.

Corollary 3.11 (Permutation bound). *Let π be any permutation of $\{1, \dots, K\}$, and define $\sigma(d_p)$ to be the set of all rule lists whose prefix starts with d_p, as in* (1). *Let $d = (d_p, \delta_p, q_0, K)$ and $D = (D_p, \Delta_p, Q_0, K)$ denote rule lists with prefixes $d_p = (p_1, \dots, p_K)$ and $D_p = (p_{\pi(1)}, \dots, p_{\pi(K)})$, respectively, i.e., the antecedents in D_p correspond to a permutation of the antecedents in d_p. If the objective lower bounds of d and D obey $b(d_p, \mathbf{x}, \mathbf{y}) \le b(D_p, \mathbf{x}, \mathbf{y})$, then the objective of the optimal rule list in $\sigma(d_p)$ gives a lower bound on the objective of the optimal rule list in $\sigma(D_p)$:*

$$\min_{d' \in \sigma(d_p)} R(d', \mathbf{x}, \mathbf{y}) \le \min_{D' \in \sigma(D_p)} R(D', \mathbf{x}, \mathbf{y}). \tag{16}$$

Thus if prefixes d_p and D_p have the same antecedents, up to a permutation, and their objective lower bounds obey $b(d_p, \mathbf{x}, \mathbf{y}) \le b(D_p, \mathbf{x}, \mathbf{y})$, Corollary 3.11 implies that we can prune D_p. We call this symmetry-aware pruning, and we illustrate the subsequent computational savings next in §3.9.1.

3.9.1 Upper bound on prefix evaluations with symmetry-aware pruning. Here, we present an upper bound on the total number of prefix evaluations that accounts for the effect of symmetry-aware pruning (§3.9). Since every subset of K antecedents generates an equivalence class of $K!$ prefixes equivalent up to permutation, symmetry-aware pruning dramatically reduces the search space.

Algorithm 1 describes a breadth-first exploration of the state space of rule lists. Now suppose we integrate symmetry-aware pruning into our execution of branch-and-bound, so that after evaluating prefixes of length K, we only keep a single best prefix from each set of prefixes equivalent up to a permutation.

Theorem 3.12 (Upper bound on prefix evaluations with symmetry-aware pruning). *Consider a state space of all rule lists formed from a set S of M antecedents, and consider the branch-and-bound algorithm with symmetry-aware pruning. Define $\Gamma_{\text{tot}}(S)$ to be the total number of prefixes evaluated. For any set S of M rules,*

$$\Gamma_{\text{tot}}(S) \le 1 + \sum_{k=1}^{K} \frac{1}{(k-1)!} \cdot \frac{M!}{(M-k)!}, \tag{17}$$

where $K = \min(\lfloor 1/2\lambda \rfloor, M)$.

Proof. By Corollary 3.4, $K \equiv \min(\lfloor 1/2\lambda \rfloor, M)$ gives an upper bound on the length of any optimal rule list. The algorithm begins by evaluating the empty prefix, followed by M prefixes of length $k = 1$, then $P(M, 2)$ prefixes of length $k = 2$, where $P(M, 2)$ is the number of size-2 subsets of $\{1, \dots, M\}$. Before proceeding to length $k = 3$, we keep only $C(M, 2)$ prefixes of length $k = 2$, where $C(M, k)$ denotes the number of k-combinations of M. Now, the number of length $k = 3$ prefixes we evaluate is $C(M, 2)(M - 2)$. Propagating this forward gives $\Gamma_{\text{tot}}(S) \le 1 + \sum_{k=1}^{K} C(M, k-1)(M - k + 1)$. □

Pruning based on permutation symmetries thus yields significant computational savings. Let us compare, for example, to the naïve number of prefix evaluations given by the upper bound in Proposition 3.7. If $M = 100$ and $K = 5$, then the naïve number is about 9.1×10^9, while the reduced number due to symmetry-aware pruning is about 3.9×10^8, which is smaller by a factor of about 23. If $M = 1000$ and $K = 10$, the number of evaluations falls from about 9.6×10^{29} to about 2.7×10^{24}, which is smaller by a factor of about 360,000. While 10^{24} seems infeasibly enormous, it does not represent the number of rule lists we evaluate. As we show in §5, our permutation bound in Corollary 3.11 and our other bounds together conspire to reduce the search space to a size manageable on a single computer. The choice of $M = 1000$ and $K = 10$ in our example above corresponds to the state space size our efforts target. $K = 10$ rules represents a (heuristic) upper limit on the size of an interpretable rule list, and $M = 1000$ represents the approximate number of rules with sufficiently high support (Theorem 3.8) we expect to obtain via rule mining (§3.1).

3.10 Equivalent points bound

The bounds in this section quantify the following: If multiple observations that are not captured by a prefix d_p have identical features and opposite labels, then no rule list that starts with d_p can correctly classify all these observations. For each set of such observations, the number of mistakes is at least the number of observations with the minority label within the set.

Consider a dataset $\{(x_n, y_n)\}_{n=1}^N$ and also a set of antecedents $\{s_m\}_{m=1}^M$. Define distinct observations to be equivalent if they are captured by exactly the same antecedents, *i.e.*, $x_i \neq x_j$ are equivalent if

$$\frac{1}{M} \sum_{m=1}^{M} \mathbb{1}[\text{cap}(x_i, s_m) = \text{cap}(x_j, s_m)] = 1. \tag{18}$$

Notice that we can partition a dataset into sets of equivalent points; let $\{e_u\}_{u=1}^U$ enumerate these sets. Let e_u be the equivalent points set that contains observation x_i. Now define $\theta(e_u)$ to be the normalized support of the minority class label with respect to set e_u, *e.g.*, let $e_u = \{x_n : \forall m \in [M],\ \mathbb{1}[\text{cap}(x_n, s_m) = \text{cap}(x_i, s_m)]\}$, and let q_u be the minority class label among points in e_u, then

$$\theta(e_u) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}[x_n \in e_u]\, \mathbb{1}[y_n = q_u]. \tag{19}$$

The existence of equivalent points sets with non-singleton support yields a tighter objective lower bound that we can combine with our other bounds; as our experiments demonstrate (§5), the practical consequences can be dramatic. First, for intuition, we present a general bound in Proposition 3.13; next, we explicitly integrate this bound into our framework in Theorem 3.14.

Proposition 3.13 (General equivalent points bound). *Let $d = (d_p, \delta_p, q_0, K)$ be a rule list, then $R(d, \mathbf{x}, \mathbf{y}) \geq \sum_{u=1}^{U} \theta(e_u) + \lambda K$.*

Now, recall that to obtain our lower bound $b(d_p, \mathbf{x}, \mathbf{y})$ in (5), we simply deleted the default rule misclassification error $\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y})$ from the objective $R(d, \mathbf{x}, \mathbf{y})$. Theorem 3.14 obtains a tighter objective lower bound via a tighter lower bound on the default rule misclassification error, $0 \leq b_0(d_p, \mathbf{x}, \mathbf{y}) \leq \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y})$.

Theorem 3.14 (Equivalent points bound). *Let d be a rule list with prefix d_p and lower bound $b(d_p, \mathbf{x}, \mathbf{y})$, then for any rule list $d' \in \sigma(d)$ whose prefix d'_p starts with d_p,*

$$R(d', \mathbf{x}, \mathbf{y}) \geq b(d_p, \mathbf{x}, \mathbf{y}) + b_0(d_p, \mathbf{x}, \mathbf{y}), \quad \textit{where} \tag{20}$$

$$b_0(d_p, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \neg\, \text{cap}(x_n, d_p) \wedge \mathbb{1}[x_n \in e_u]\, \mathbb{1}[y_n = q_u].$$

4 IMPLEMENTATION

For every prefix d_p evaluated during Algorithm 1's execution, we compute the objective lower bound $b(d_p, \mathbf{x}, \mathbf{y})$ and sometimes the objective $R(d, \mathbf{x}, \mathbf{y})$ of the corresponding rule list d. These calculations are the dominant computations and motivate our use of a highly optimized library, designed by Yang et al. [45], for representing rule lists and performing operations encountered in evaluating functions of rule lists. Furthermore, we exploit the hierarchical nature of the objective function and its lower bound to compute these quantities incrementally throughout branch-and-bound execution.

We implement our algorithm using a collection of optimized data structures: a trie (prefix tree), a symmetry-aware map, and a queue. The trie acts like a cache, keeping track of prefixes that we have already evaluated and are also still of interest. Each node in the trie contains metadata associated with that corresponding rule list; the metadata consists of bookkeeping information such as what child rule lists are feasible and the lower bound and accuracy for that rule list. We also track the best observed minimum objective and its associated rule list.

The symmetry-aware map supports symmetry-aware pruning. We implement this using the C++ STL unordered_map, to map all permutations of a set of antecedents to a key, whose value contains the best ordering of those antecedents (*i.e.*, the prefix with the smallest lower bound). Every antecedent is associated with an index, and we call the numerically sorted order of a set of antecedents its canonical order. Thus by querying a set of antecedents by its canonical order, all permutations map to the same key. This map dominates memory usage for problems that explore longer prefixes. Before inserting permutation P_i into the symmetry-aware map, we check if there exists a permutation P_j of P_i already in the map. If no such permutation exists, then we insert P_i in the map. Otherwise, if a permutation P_j exists and the lower bound of P_i is better than that of P_j, we update the map and remove P_j and its subtree from the trie. Else, if P_j exists and has a better lower bound than P_i, we do nothing (*i.e.*, we do not insert P_i into the symmetry-aware map or the trie).

We use a queue to store all of the leaves of the trie that still need to be explored. We order entries in the queue to implement several different policies, including breadth-first search (BFS) and best-first search. For best-first we use a priority queue, ordered by either the lower bound, the objective, or a custom priority metric. In preliminary work (not shown), we also experimented with stochastic exploration processes that do not require a queue; instead, these follow random paths from the root to leaves. Developing such methods could be an interesting direction for future work. We find that ordering by the lower bound and other priority metrics often leads to a shorter runtime than using BFS.

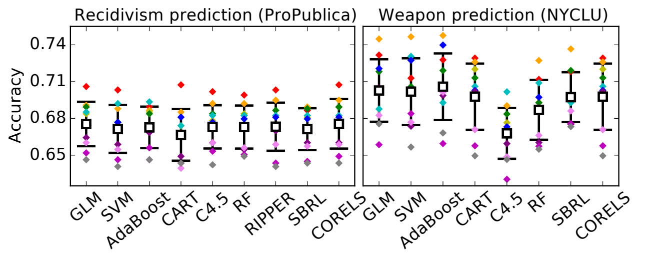

Figure 2: Test accuracy means (white squares), standard deviations (error bars), and values (colors correspond to folds). For CORELS, $\lambda = 0.005$ (left) and $\lambda = 0.01$ (right).

Mapping our algorithm to our data structures produces the following execution strategy. While the trie contains unexplored leaves, a scheduling policy selects the next prefix to extend. Then, for every antecedent that is not already in this prefix, we calculate the lower bound, objective, and other metrics for the rule list formed by appending the antecedent to the prefix. If the lower bound of the new rule list is less than the current minimum objective, we insert that rule list into the symmetry-aware map, trie, and queue, and, if relevant, update the current minimum objective. If the lower bound is greater than the minimum objective, then no extension of this rule list could possibly be optimal, thus we do not insert the new rule list into the tree or queue. We also leverage our other bounds from §3 to aggressively prune the search space.

During execution, we garbage collect the trie. Each time we update the minimum objective, we traverse the trie in a depth-first manner, deleting all subtrees of any node with lower bound larger than the current minimum objective. At other times, when we encounter a node with no children, we prune upwards, deleting that node and recursively traversing the tree towards the root, deleting any childless nodes. This garbage collection allows us to constrain the trie's memory consumption, though in our experiments we observe the minimum objective to decrease only a few times.

5 EXPERIMENTS

Our experimental analysis addresses four questions: (1) How does CORELS' accuracy and model size compare to that of other algorithms? (2) How rapidly do the objective value and its lower bound converge, for different values of λ? (3) How much does each of the implementation optimizations contribute to CORELS' performance? (4) How rapidly does CORELS prune the search space? We present additional empirical results and further implementation and data processing details in a long version of this report [1].

All timed results ran on a server with two Intel Xeon E5-2699 v4 (55 MB cache, 2.20 GHz) processors and 756 GB RAM. Except where we mention a memory constraint, all experiments can run comfortably on smaller machines, *e.g.*, a laptop with 16GB RAM.

We focus on two problems: (1) Predicting which individuals in the ProPublica COMPAS dataset [23] recidivate within two years ($N = 7,215$). Our rule mining procedure yields $M = 156$ single- and two-clause antecedents. (2) Using the NYCLU 2014 stop-and-frisk dataset [30] to predict whether a weapon will be found on a stopped individual who is frisked or searched. We identify a subset of 29,595 records for stopped individuals who were frisked and/or searched. Of these, criminal possession of a weapon was identified in about 5% of instances, thus we resampled due to class imbalance. In the first

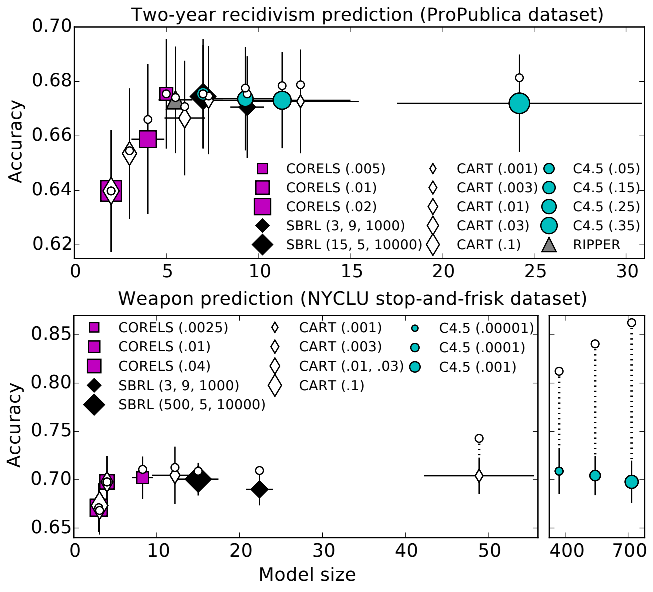

Figure 3: Training and test accuracy as a function of model size. In the legend, numbers in parentheses are algorithm parameters that we vary for CORELS (λ), CART (cp), C4.5 (C), and SBRL (η, λ, i), where i is the number of iterations. Legend markers and error bars indicate means and standard deviations, respectively, of test accuracy across cross-validation folds. Small circles mark training accuracy means. Top: ProPublica dataset. No models exhibit significant overfitting. Bottom: NYCLU dataset. CART with $cp = 0.001$ overfits; C4.5 finds large models and dramatically overfits. Note the broken horizontal axis and change in scale.

two subsections below, we use $M = 28$ single-clause antecedents; in the latter two, we add negations of some, yielding $M = 46$.

Accuracy and model size: We first ran a 10-fold cross validation experiment using CORELS and eight other algorithms:[3] logistic regression, support vector machines, AdaBoost, CART, C4.5, random forests, RIPPER,[4] and scalable Bayesian rule lists (SBRL).[5] Figure 1 shows an optimal rule list that CORELS learns for the ProPublica dataset. Figure 2 shows that there were no statistically significant differences in algorithm accuracies – the difference between folds was far larger than the difference between algorithms. We conclude that CORELS produces models whose accuracy is comparable to those found via other algorithms. Figure 3 summarizes differences in accuracy and model size for CORELS and other tree (CART, C4.5) and rule list (RIPPER, SBRL) learning algorithms. For both problems, CORELS can learn short rule lists without sacrificing accuracy.

Convergence and regularization: We illustrate several views of CORELS execution traces, for the NYCLU stop-and-frisk dataset with $M = 28$ antecedents, for the same three regularization parameters ($\lambda = 0.04, 0.01, 0.025$) as in Figure 3 (bottom). The panels in Figure 4 plot example execution traces, from a single cross-validation fold, of both the current best objective value R^c and the lower

[3]We use standard R packages, with default parameter settings, for the first seven. By default, CART and C4.5 use complexity parameters $cp = 0.01$ and $C = 0.25$, respectively.
[4]We were unable to execute RIPPER for the NYCLU problem.
[5]Code for SBRL can be found at https://github.com/Hongyuy/sbrlmod. By default, SBRL sets $\eta = 3$, $\lambda = 9$, the number of chains to 11 and iterations to 1,000.

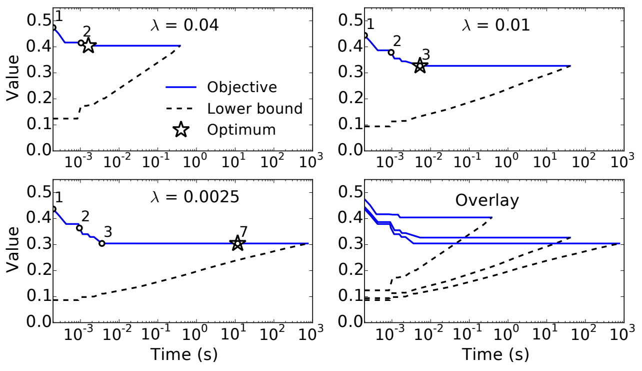

Figure 4: Example CORELS execution traces (NYCLU). Objective value (lines) and lower bound (dashes) for CORELS. Numbered points along the trace of the objective value indicate when the length of the best known rule list changes and are labeled by the new length. For each value of λ, a star marks the optimum and the time at which it was achieved. Bottom right: Overlay of the three traces.

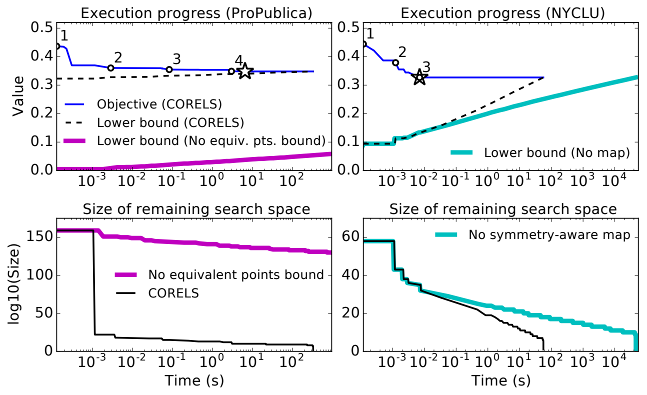

Figure 5: Significant algorithm optimizations for ProPublica (left) and NYCLU (right). Top: Objective value (thin lines) and lower bound (dashes) for CORELS, as in Figure 4, and lower bound (thick lines) for separate executions of variants without equivalent points (left) and permutation (right) bounds. Bottom: Upper bound on remaining search space size, for CORELS (thin lines) and variants (thick lines).

bound $b(d_p, \mathbf{x}, \mathbf{y})$ of the prefix d_p being evaluated. These plots illustrate that CORELS certifies optimality when the lower bound matches the objective value. In these examples, CORELS achieves the optimum in a small fraction of the total execution time.

As λ decreases, these times increase because the search problems become more difficult. In particular, CORELS must evaluate longer prefixes: across 10 cross-validation folds, the maximum evaluated prefix lengths are 6, 11, and 16-17, for $\lambda = 0.04$, 0.01, and 0.025, respectively. Consequently, our data structures grow in size: *e.g.*, across folds, the mean number of queue elements are 1,900 (100), 170,000 (1,300), and 2,400,000 (170,000), with standard deviations shown in parentheses, for $\lambda = 0.04$, 0.01, and 0.025, respectively.

Algorithm optimizations: We determine the efficacy of each of our bounds and data structure optimizations. In the remainder, we show results using the ProPublica dataset. Table 1 provides summary

Algorithm variant	t_{total} (min)	t_{opt} (s)	i_{total} ($\times 10^6$)	$Q_{\max}$ ($\times 10^6$)	$K_{\max}$
CORELS	5.4 (1.6)	8 (2)	1.7 (0.4)	1.3 (0.4)	5-6
BFS priority queue	6.4 (2.0)	34 (21)	3.5 (1.4)	2.6 (1.2)	5-6
No support bounds	10.0 (3.3)	12 (4)	2.7 (0.8)	2.2 (0.7)	5-6
No symmetry-aware map	56.8 (22.3)	21 (6)	16.0 (5.9)	14.5 (5.7)	5-6
No lookahead bound	70.0 (22.0)	8 (2)	18.5 (5.9)	17.1 (5.5)	6-7
No equivalent pts bound	>87	>3744	>1004	>987	≥10

Table 1: Per-component performance improvement (ProPublica). Columns report total execution time, time to optimum, number of queue insertions, maximum queue size, and maximum evaluated prefix length. The first row shows CORELS; subsequent rows show variants that each remove one implementation optimization or bound. We terminated each experiment in the last row once the size of the trie reached 10^9 nodes (each execution consumed ~350GB RAM). In all but the final row and column, we report means (and standard deviations) over 10 cross-validation folds; in the final row, we report the minimum values across folds.

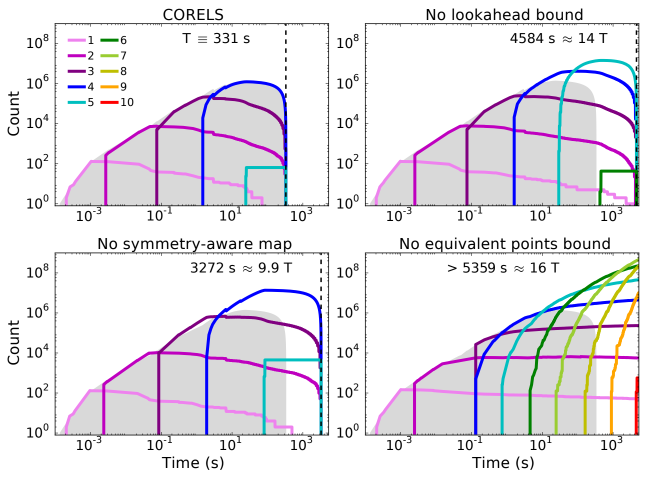

Figure 6: Queue composition (ProPublica). Numbers of prefixes in the queue (log scale), labeled and colored by length, for CORELS (top left) and without three specific implementation optimizations or bounds. Gray shades the area beneath the total number of queue elements for CORELS, *i.e.*, the sum over all lengths in the top left figure. For comparison, we replicate the same gray region in all subfigures.

statistics for experiments using the full CORELS implementation and variants that each remove a specific optimization. Figure 6 presents a view of the same experiments, focusing on three of our optimizations. These plots depict the number of prefixes of a given length in the queue during the algorithm's execution.

State space pruning: Figure 5 highlights the most significant algorithm optimizations for our two problems: the equivalent points bound for the ProPublica dataset (left) and the symmetry-aware map for the NYCLU dataset (right). On the left, for CORELS (thin lines), the objective drops quickly, achieving the optimal value within 10 seconds. CORELS certifies optimality in less than 6 minutes: the objective lower bound steadily converges to the optimal objective (top) as the search space shrinks (bottom). We dynamically and incrementally calculate $\lfloor \log_{10} \Gamma(R^c, Q) \rfloor$, where $\Gamma(R^c, Q)$

is the upper bound on remaining search space size (Theorem 3.6); this adds some computational overhead. In the same plots, we also show a separate execution of CORELS without the equivalent points bound (Theorem 3.14). This execution is far from complete: the lower bound is far from the optimum objective value (top) and much of the search space remains unexplored (bottom). On the right, CORELS achieves the optimum objective in well under a second and certifies optimality in less than a minute. Removing the permutation bound (Corollary 3.11), and thus the symmetry-aware map, increases the execution time by orders of magnitude.

6 CONCLUSION

CORELS is an efficient and accurate algorithm for constructing provably optimal rule lists. Optimality is particularly important in domains where model interpretability has social consequences, *e.g.*, recidivism prediction. While achieving optimality on such discrete optimization problems is computationally hard in general, we aggressively prune our problem's search space via a suite of bounds. This makes realistically sized problems tractable. CORELS is amenable to parallelization, which should allow it to scale to even larger problems.

ACKNOWLEDGEMENTS

E.A. is supported by the Miller Institute for Basic Research in Science, University of California, Berkeley, and is hosted by Prof. M.I. Jordan at RISELab. C.D.R. is supported in part by MIT-Lincoln Labs.

REFERENCES

[1] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin. 2017. Learning certifiably optimal rule lists for categorical data. *Preprint at arXiv:1704.01701* (April 2017).
[2] K. P. Bennett and J. A. Blue. 1996. *Optimal Decision Trees.* Technical Report. R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute.
[3] I. Bratko. 1997. Machine learning: Between accuracy and interpretability. In *Learning, Networks and Statistics.* International Centre for Mechanical Sciences, Vol. 382. Springer Vienna, 163–177.
[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Wadsworth.
[5] C. Chen and C. Rudin. 2017. Optimized Falling Rule Lists and Softly Falling Rule Lists. (2017). Work in progress.
[6] H. A. Chipman, E. I. George, and R. E. McCulloch. 1998. Bayesian CART model search. *J. Amer. Statist. Assoc.* 93, 443 (1998), 935–948.
[7] H. A. Chipman, E. I. George, and R. E. McCulloch. 2002. Bayesian treed models. *Machine Learning* 48, 1 (2002), 299–320.
[8] H. A. Chipman, E. I. George, and R. E. McCulloch. 2010. BART: Bayesian additive regression trees. *The Annals of Applied Statistics* 4, 1 (2010), 266–298.
[9] P. Clark and T. Niblett. 1989. The CN2 induction algorithm. *Machine Learning* 3 (1989), 261–283. Issue 4.
[10] W. W. Cohen. 1995. Fast Effective Rule Induction. In *Twelfth International Conference on Machine Learning (ICML).* 115–123.
[11] R. M. Dawes. 1979. The robust beauty of improper linear models in decision making. *American Psychologist* 34, 7 (1979), 571–582.
[12] D. Dension, B. Mallick, and A.F.M. Smith. 1998. A Bayesian CART algorithm. *Biometrika* 85, 2 (1998), 363–377.
[13] D. Dobkin, T. Fulton, D. Gunopulos, S. Kasif, and S. Salzberg. 1996. Induction of shallow decision trees. (1996).
[14] A. Farhangfar, R. Greiner, and M. Zinkevich. 2008. A Fast Way to Produce Optimal Fixed-Depth Decision Trees. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2008).*
[15] Eibe Frank and Ian H. Witten. 1998. Generating Accurate Rule Sets Without Global Optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98).* 144–151.
[16] A. A. Freitas. 2014. Comprehensible classification models: a position paper. *ACM SIGKDD Explorations Newsletter* 15, 1 (2014), 1–10.
[17] M. Garofalakis, D. Hyun, R. Rastogi, and K. Shim. 2000. Efficient Algorithms for Constructing Decision Trees with Constraints. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'98).* 335–339.
[18] C. Giraud-Carrier. 1998. Beyond predictive accuracy: What?. In *Proceedings of the ECML-98 Workshop on Upgrading Learning to Meta-Level: Model Selection and Data Transformation.* 78–85.
[19] B. Goodman and S. Flaxman. 2016. European Union regulations on algorithmic decision-making and a "right to explanation". In *ICML Workshop on Human Interpretability in Machine Learning (WHI).*
[20] R. C. Holte. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine Learning* 11, 1 (1993), 63–91.
[21] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. 2011. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems* 51, 1 (2011), 141–154.
[22] H. Lakkaraju and C. Rudin. 2017. Cost-sensitive and Interpretable Dynamic Treatment Regimes Based on Rule Lists. In *Proceedings of the Artificial Intelligence and Statistics (AISTATS).*
[23] J. Larson, S. Mattu, L. Kirchner, and J. Angwin. 2016. How We Analyzed the COMPAS Recidivism Algorithm. *ProPublica* (2016).
[24] N. L. Larus-Stone. 2017. *Learning Certifiably Optimal Rule Lists: A Case For Discrete Optimization in the 21st Century.* (2017). Undergraduate thesis, Harvard College.
[25] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. 2015. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics* 9, 3 (2015), 1350–1371.
[26] W. Li, J. Han, and J. Pei. 2001. CMAR: Accurate and efficient classification based on multiple class-association rules. *IEEE International Conference on Data Mining* (2001), 369–376.
[27] B. Liu, W. Hsu, and Y. Ma. 1998. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD '98).* 80–96.
[28] M. Marchand and M. Sokolova. 2005. Learning with decision lists of data-dependent features. *Journal of Machine Learning Research* 6 (2005), 427–451.
[29] R.S. Michalski. 1969. On the quasi-minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing.* 125–128.
[30] New York Civil Liberties Union. 2014. Stop-and-Frisk Data. (2014). http://www.nyclu.org/content/stop-and-frisk-data.
[31] S. Nijssen and E. Fromont. 2010. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery* 21, 1 (2010), 9–51.
[32] J. R. Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann.
[33] P. R. Rijnbeek and J. A. Kors. 2010. Finding a Short and Accurate Decision Rule in Disjunctive Normal Form by Exhaustive Search. *Machine Learning* 80, 1 (July 2010), 33–62.
[34] R. L. Rivest. 1987. Learning Decision Lists. *Machine Learning* 2, 3 (Nov. 1987), 229–246.
[35] U. Rückert and L. De Raedt. 2008. An experimental evaluation of simplicity in rule learning. *Artificial Intelligence* 172 (2008), 19–28.
[36] C. Rudin and S. Ertekin. 2015. Learning Optimized Lists of Rules with Mathematical Programming. (2015). Unpublished.
[37] C. Rudin, B. Letham, and D. Madigan. 2013. Learning Theory Analysis for Association Rules and Sequential Event Prediction. *Journal of Machine Learning Research* 14 (2013), 3384–3436.
[38] S. Rüping. 2006. *Learning interpretable models.* Ph.D. Dissertation. Universität Dortmund.
[39] G. Shmueli. 2010. To explain or to predict? *Statist. Sci.* 25, 3 (Aug. 2010), 289–310.
[40] M. Sokolova, M. Marchand, N. Japkowicz, and J. Shawe-Taylor. 2003. The Decision List Machine. In *Advances in Neural Information Processing Systems (NIPS '03),* Vol. 15. 921–928.
[41] K. Vanhoof and B. Depaire. 2010. Structure of association rule classifiers: A review. In *Proceedings of the International Conference on Intelligent Systems and Knowledge Engineering (ISKE '10).* 9–12.
[42] A. Vellido, J. D. Martín-Guerrero, and P. J.G. Lisboa. 2012. Making machine learning models interpretable. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning.*
[43] F. Wang and C. Rudin. 2015. Causal Falling Rule Lists. *Preprint at arXiv:1510.05189* (Oct. 2015).
[44] F. Wang and C. Rudin. 2015. Falling Rule Lists. In *Proceedings of Artificial Intelligence and Statistics (AISTATS).*
[45] H. Yang, C. Rudin, and M. Seltzer. 2017. Scalable Bayesian Rule Lists. In *Proceedings of the 34th International Conference on Machine Learning (ICML '17).*
[46] X. Yin and J. Han. 2003. CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining (ICDM '03).* 331–335.
[47] Y. Zhang, E. B. Laber, A. Tsiatis, and M. Davidian. 2015. Using decision lists to construct interpretable and parsimonious treatment regimes. *Biometrics* 71, 4 (2015), 895–904.