Learning Certifiably Optimal Rule Lists for Categorical Data

Elaine Angelino ELAINE@EECS.BERKELEY.EDU
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, Berkeley, CA 94720

Nicholas Larus-Stone NLARUSSTONE@COLLEGE.HARVARD.EDU
Daniel Alabi ALABID@G.HARVARD.EDU
Margo Seltzer MARGO@EECS.HARVARD.EDU
School of Engineering and Applied Sciences
Harvard University, Cambridge, MA 02138

Cynthia Rudin CYNTHIA@CS.DUKE.EDU
Department of Computer Science and Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708

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.

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 (Rivest, 1987; Breiman et al., 1984; Quinlan, 1993). Recent approaches use Bayesian analysis, either to find a locally optimal solution (Chipman et al., 1998) or to explore the search space (Letham et al., 2015; Yang et al., 2016). 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.

Optimality is important, because there are societal implications for a lack of optimality. Consider the recent ProPublica article on the COMPAS recidivism prediction tool (Larson et al., 2016). 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

if $(age = 23 - 25) \wedge (priors = 2 - 3)$ **then predict** *yes*
else if $(age = 18 - 20)$ **then predict** *yes*
else if $(sex = male) \wedge (age = 21 - 22)$ **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.

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 (Larson et al., 2016), 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. 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.

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 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.

Within its branch-and-bound procedure, CORELS maintains a lower bound on the minimum value of R that each incomplete rule list can achieve. This allows CORELS to prune an incomplete rule list (and every possible extension) if the bound is larger than the error of the best rule list that it has already evaluated. The use of careful bounding techniques leads to massive pruning of the search space of potential rule lists. It continues to consider incomplete and complete rule lists until it has either examined or eliminated every rule list from consideration. Thus, CORELS terminates with the optimal rule list and a certificate of optimality.

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 allow 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, and is parallelizable.

We evaluated CORELS on a number of publicly available datasets. Our metric of success was 10-fold cross-validated prediction accuracy on a subset of the data. These datasets involve hundreds of rules and thousands of observations. CORELS is generally able to find an optimal rule list in a matter of seconds and certify its optimality within about 10 minutes. We show that we are able to achieve better or similar out-of-sample accuracy on these datasets compared to the popular greedy algorithms, CART and C4.5.

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 (Larson et al., 2016), for the problem of two-year recidivism prediction, and (2) the NYCLU 2014 stop-and-frisk dataset (New York Civil Liberties Union, 2014), 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 implementation of CORELS is at `https://github.com/nlarusstone/corels`. [1]

2. Related Work

We discuss related literature in several subfields, and highlight two recent works that this paper builds on.

Interpretable Models: There is a growing interest in interpretable (transparent, comprehensible) models because of their societal importance (see Rüping, 2006; Bratko, 1997;

1. Our work overlaps with the thesis presented by Larus-Stone (2017).

Dawes, 1979; Vellido et al., 2012; Giraud-Carrier, 1998; Holte, 1993; Shmueli, 2010; Huysmans et al., 2011; Freitas, 2014). There are now regulations on algorithmic decision-making in the European Union on the "right to an explanation" (Goodman and Flaxman, 2016) that would legally require interpretability in predictions.

Optimal Decision Tree Modeling: The body of work closest to ours is possibly that of optimal decision tree modeling. Since the late 1990's, there has been research on building optimal decision trees using optimization techniques (Bennett and Blue, 1996; Dobkin et al., 1996), continuing until the present (Farhangfar et al., 2008). A particularly interesting paper along these lines is that of Nijssen and Fromont (2010), 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. [2] Another work close to ours is that of Garofalakis et al. (2000), who introduce an algorithm to generate more interpretable decision trees by allowing constraints to be placed on the size of the decision tree. During tree construction, they bound the possible Minimum Description Length (MDL) cost of every different split at a given node. If every split at that node is more expensive than the actual cost of the current subtree, then that node can be pruned. In this way, they are able to prune the tree while constructing it instead of just constructing the tree and then pruning at the end. They do not aim for optimal trees; they build trees that obey constraints, and find optimal subtrees within the trees that were built during the building phase.

Greedy splitting and pruning: Unlike optimal decision tree methods, methods like CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993) do not perform exploration of the search space beyond greedy splitting. There are a huge number of algorithms in this class.

Bayesian tree and rule list methods: Some of these approaches that aim to explore the space of trees (Dension et al., 1998; Chipman et al., 2002, 2010) use Monte Carlo methods. However, the space of trees of a given depth is much larger than the space of rule lists of that same level of depth, and the trees within these algorithms are grown in a top-down greedy way. Because of this, the authors noted that their MCMC chains tend to reach only locally optimal solutions. This explains why Bayesian rule-based methods (Letham et al., 2015; Yang et al., 2016) have tended to be more successful in escaping local minima. Our work builds specifically on that of Yang et al. (2016). In particular, we use their library for efficiently representing and operating on bit vectors, and build on their bounds. Note that the RIPPER algorithm (Cohen, 1995) is similar to the Bayesian tree methods in that it grows, prunes, and then locally optimizes.

Rule learning methods: Most rule learning methods are not designed for optimality or interpretability, but for computational speed and/or accuracy. In *associative classification* (Vanhoof and Depaire, 2010; Liu et al., 1998; Li et al., 2001; Yin and Han, 2003), classifiers are often formed greedily from the top down as rule lists, or they are formed by taking the simple union of pre-mined rules, whereby any observation that fits into any of the rules is classified as positive. In *inductive logic programming* (Muggleton and De Raedt, 1994), algorithms construct disjunctive normal form patterns via a set of operations (rather than using optimization). These approaches are not appropriate for obtaining a guarantee of

2. There is no public version of their code for distribution as of this writing.

optimality. Methods for decision list learning construct rule lists iteratively in a greedy way (Rivest, 1987; Sokolova et al., 2003; Marchand and Sokolova, 2005; Rudin et al., 2013; Goessling and Kang, 2015); these too have no guarantee of optimality, and tend not to produce optimal rule lists in general. Some methods allow for interpretations of single rules, without constructing rule lists (McCormick et al., 2012).

There is a tremendous amount of related work in other subfields that are too numerous to discuss at length here. We have not discussed *rule mining* algorithms since they are part of an interchangeable preprocessing step for our algorithm and are deterministically fast (*i.e.*, they will not generally slow our algorithm down). We also did not discuss methods that create disjunctive normal form models, *e.g.*, logical analysis of data, and many associative classification methods.

Related problems with interpretable lists of rules: Beyond trees that are optimized for accuracy and sparsity, rule lists have been developed for various applications, and with exotic types of constraints. For example, Falling Rule Lists (Wang and Rudin, 2015) are constrained to have decreasing probabilities down the list as are rule lists for dynamic treatment regimes (Zhang et al., 2015) and cost-sensitive dynamic treatment regimes (Lakkaraju and Rudin, 2017). Both Wang and Rudin (2015) and Lakkaraju and Rudin (2017) use Monte Carlo searches to explore the space of rule lists. The method proposed in this paper could potentially be adapted to handle these kinds of interesting problems. We are currently working on bounds for Falling Rule Lists (Chen and Rudin, 2017) similar to those presented here.

Two works that this paper builds on are those of Yang et al. (2016), and Rudin and Ertekin (2015). The work of Yang et al. provided the bit vector libraries and several ideas that were used here, and we used their code as a starting point. Their scalable Bayesian rule lists (SBRL) method has uses beyond those of CORELS because SBRL models are probabilistic, producing an estimate of $\mathbb{P}(Y = 1 \,|\, X)$ for any X, rather than a yes/no classification. On the other hand, because the model is probabilistic, the bounds depend on approximations involving gamma functions. Bounds for CORELS have no such approximations and are substantially tighter. Yang et al. aim to find the optimal solution but do not aim to prove optimality.

Both Yang et al. (2016) and Rudin and Ertekin (2015) contributed bounds that we started from in this work, and in particular, the latter uses the same objective as we do and has some of the same bounds, including the minimum support bound (§3.7, Theorem 10). However, Ertekin and Rudin's work is for a different purpose, namely it is for building rule lists that can be customized; since the authors use mixed integer programming (MIP), users can easily add constraints to the MIP and create rule lists that obey these arbitrary constraints. As in our framework, their rule lists are certifiably optimal. However, since generic MIP software is used without efficient bounds and data structures like the ones introduced here, much more time can be required to prove optimality and to find the optimal solution.

3. Learning optimal rule lists

In this section, we present our framework for learning certifiably optimal rule lists. First, we define our setting and useful notation (§3.1), followed by the objective function we seek to minimize (§3.2). Next, we describe the principal structure of our optimization algo-

if $(age = 23-25) \wedge (priors = 2-3)$ **then predict** yes	**if** p_1 **then predict** q_1
else if $(age = 18-20)$ **then predict** yes	**else if** p_2 **then predict** q_2
else if $(sex = male) \wedge (age = 21-22)$ **then predict** yes	**else if** p_3 **then predict** q_3
else if $(priors > 3)$ **then predict** yes	**else if** p_4 **then predict** q_4
else predict no	**else predict** q_0

Figure 2: The same 4-rule list $d = (r_1, r_2, r_3, r_4, r_0)$, as in Figure 1, that predicts two-year recidivism for the ProPublica dataset. Each rule is of the form $r_k = p_k \to q_k$, for all $k = 0, \ldots, 4$. We also equivalently 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 $K = 4$.

rithm (§3.3), which depends on a hierarchically structured objective lower bound (§3.4). We then derive a series of additional bounds that we incorporate into our algorithm because they enable aggressive pruning of our state space.

3.1 Rule lists for binary classification

We restrict our setting to binary classification, where rule lists are Boolean functions; this framework is straightforward to generalize to multi-class 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 $d = (r_1, r_2, \ldots, r_K, r_0)$ of length $K \geq 0$ is a $(K+1)$-tuple consisting of K distinct association rules, $r_k = p_k \to q_k$, for $k = 1, \ldots, K$, followed by a default rule r_0. Figure 2 illustrates a rule list, which for clarity, we sometimes call a K-rule list. 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 special association rule $p_0 \to q_0$ whose antecedent p_0 simply asserts true.

Let $d = (r_1, r_2, \ldots, r_K, r_0)$ be a K-rule list, where $r_k = p_k \to q_k$ for each $k = 0, \ldots, K$. We introduce a useful alternate rule list representation: $d = (d_p, \delta_p, q_0, K)$, where we define $d_p = (p_1, \ldots, p_K)$ to be d's prefix, $\delta_p = (q_1, \ldots, q_K) \in \{0,1\}^K$ gives the label predictions associated with d_p, and $q_0 \in \{0,1\}$ is the default label prediction. In Figure 1, $d = (r_1, r_2, r_3, r_4, r_0)$, and each rule is of the form $r_k = p_k \to q_k$, for all $k = 0, \ldots, 4$; equivalently, $d = (d_p, \delta_p, q_0, K)$, where $d_p = (p_1, p_2, p_3, p_4)$, $\delta_p = (1, 1, 1, 1)$, $q_0 = 0$, and $K = 4$.

Let $d_p = (p_1, \ldots, p_k, \ldots, p_K)$ be an antecedent list, then for any $k \leq K$, we define $d_p^k = (p_1, \ldots, p_k)$ 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:

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

If $d_p = (p_1, \ldots, p_K)$ and $d'_p = (p_1, \ldots, p_K, p_{K+1})$ 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. We also say that 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:

$$\{x_n : \text{cap}(x_n, d_p)\} \subseteq \{x_n : \text{cap}(x_n, d'_p)\}. \tag{2}$$

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 $d_p = (p_1, \ldots, p_k, \ldots, p_K)$ is a prefix, then

$$\text{cap}(x_n, p_k \,|\, d_p) = \left(\bigwedge_{k'=1}^{k-1} \neg \, \text{cap}(x_n, p_{k'}) \right) \wedge \text{cap}(x_n, p_k) \tag{3}$$

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,

$$\text{supp}(\beta, \mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, \beta), \tag{4}$$

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

$$\text{supp}(\beta, \mathbf{x} \,|\, d_p) = \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, \beta \,|\, d_p), \tag{5}$$

Next, we address how empirical data constrains rule lists. Given training data $(\mathbf{x}, \mathbf{y})$, an antecedent list $d_p = (p_1, \ldots, p_K)$ implies a rule list $d = (d_p, \delta_p, q_0, K)$ with prefix d_p, where the label predictions $\delta_p = (q_1, \ldots, q_K)$ 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.

Finally, we note that our approach leverages pre-mined rules, following the methodology taken by Letham et al. (2015) and Yang et al. (2016). One of the results we later prove implies a constraint that can be used as a filter during rule mining – antecedents must have at least some minimum support given by the lower bound in Theorem 10.

3.2 Objective function

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

$$R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda K. \tag{6}$$

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 association rule.

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 that each translates into a useful bound, and that together interact to eliminate large parts of the search space. We will discuss these in depth throughout the following sections:

- Lower bounds on a prefix also hold for every extension of that prefix. (§3.4, Theorem 1)
- We can sometimes prune all rule lists that are longer than a given prefix, even without knowing anything about what rules will be placed below that prefix. (§3.4, Lemma 2)
- We can calculate *a priori* an upper bound on the maximum length of an optimal rule list. (§3.5, Theorem 6)
- 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 10)
- 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 11)
- We need only consider the optimal permutation of antecedents in a prefix; we can omit all other permutations. (§3.10, Theorem 17 and Corollary 18)
- 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. (§3.14, Theorem 22)

3.4 Hierarchical objective lower bound

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

$$\ell(d, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}), \tag{7}$$

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

$$\ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \operatorname{cap}(x_n, p_k \,|\, d_p) \wedge \mathbb{1}[q_k \neq y_n] \tag{8}$$

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

$$\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[q_0 \neq y_n] \tag{9}$$

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,

$$b(d_p, \mathbf{x}, \mathbf{y}) \equiv \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K \leq R(d, \mathbf{x}, \mathbf{y}), \tag{10}$$

where we have suppressed the lower bound's dependence on label predictions δ_p because they are fully determined, given $(d_p, \mathbf{x}, \mathbf{y})$. Furthermore, as we state next in Theorem 1, $b(d_p, \mathbf{x}, \mathbf{y})$ gives a lower bound on the objective of *any* rule list whose prefix starts with d_p.

Theorem 1 (Hierarchical objective lower bound) *Define $b(d_p, \mathbf{x}, \mathbf{y})$ as in (10). 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 $b(d_p, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y})$.*

Proof Let $d_p = (p_1, \ldots, p_K)$ and $\delta_p = (q_1, \ldots, q_K)$; let $d'_p = (p_1, \ldots, p_K, p_{K+1}, \ldots, p_{K'})$ and $\delta'_p = (q_1, \ldots, q_K, q_{K+1}, \ldots, q_{K'})$. Notice that d'_p yields the same mistakes as d_p, and possibly additional mistakes:

$$\begin{aligned}
&\ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] \\
&= \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{k=1}^{K} \operatorname{cap}(x_n, p_k \,|\, d_p) \wedge \mathbb{1}[q_k \neq y_n] + \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] \right) \\
&\geq \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}),
\end{aligned} \tag{11}$$

where we have used the fact that $\operatorname{cap}(x_n, p_k \,|\, d'_p) = \operatorname{cap}(x_n, p_k \,|\, d_p)$ for $1 \leq k \leq K$. It follows that

$$\begin{aligned}
b(d_p, \mathbf{x}, \mathbf{y}) &= \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K \\
&\leq \ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) + \lambda K' = b(d'_p, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y}).
\end{aligned} \tag{12}$$

■

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

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) ← (d^0, R^0)                    ▷ Initialize best rule list and objective
Q ← queue( [ ( ) ] )                       ▷ Initialize queue with empty prefix
while Q not empty do                       ▷ Stop when queue is empty
    d_p ← Q.pop( )                         ▷ Remove prefix d_p from the queue
    if b(d_p, x, y) < R^c then             ▷ Bound: Apply Theorem 1
        R ← R(d, x, 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) ← (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*) ← (d^c, R^c)                      ▷ Identify provably optimal solution
```

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 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 1, we needn't 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 1.

Lemma 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 $R(d', \mathbf{x}, \mathbf{y}) \geq R^c$.*

Proof By the definition of the lower bound (10), which includes the penalty for longer prefixes,

$$\begin{aligned} R(d'_p, \mathbf{x}, y) \geq b(d'_p, \mathbf{x}, \mathbf{y}) &= \ell_p(d'_p, \mathbf{x}, \mathbf{y}) + \lambda K' \\ &= \ell_p(d'_p, \mathbf{x}, \mathbf{y}) + \lambda K + \lambda(K' - K) \\ &= b(d_p, \mathbf{x}, \mathbf{y}) + \lambda(K' - K) \geq b(d_p, \mathbf{x}, \mathbf{y}) + \lambda \geq R^c. \end{aligned} \tag{13}$$

■

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

3.5 Upper bounds on prefix length

The simplest upper bound on prefix length is given by the total number of available antecedents.

Proposition 3 (Trivial upper bound on prefix length) *Consider a state space of all rule lists formed from a set of M antecedents, and let $L(d)$ be the length of rule list d. M provides an upper bound on the length of any optimal rule list $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, i.e., $L(d) \leq M$.*

Proof Rule lists consist of distinct rules by definition. ■

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 4 (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 $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$*

$$L(d^*) \leq \min\left(\left\lfloor \frac{R^c}{\lambda} \right\rfloor, M\right), \tag{14}$$

where λ is the regularization parameter. Furthermore, if d^c is a rule list with objective $R(d^c, \mathbf{x}, \mathbf{y}) = R^c$, length K, and zero misclassification error, then for every optimal rule list $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, if $d^c \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, then $L(d^*) \leq K$, or otherwise if $d^c \notin \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, then $L(d^*) \leq K - 1$.

Proof For an optimal rule list d^* with objective R^*,

$$\lambda L(d^*) \leq R^* = R(d^*, \mathbf{x}, \mathbf{y}) = \ell(d^*, \mathbf{x}, \mathbf{y}) + \lambda L(d^*) \leq R^c. \tag{15}$$

The maximum possible length for d^* occurs when $\ell(d^*, \mathbf{x}, \mathbf{y})$ is minimized; combining with Proposition 3 gives bound (14).

For the rest of the proof, let $K^* = L(d^*)$ be the length of d^*. If the current best rule list d^c has zero misclassification error, then

$$\lambda K^* \leq \ell(d^*, \mathbf{x}, \mathbf{y}) + \lambda K^* = R(d^*, \mathbf{x}, \mathbf{y}) \leq R^c = R(d^c, \mathbf{x}, \mathbf{y}) = \lambda K, \tag{16}$$

and thus $K^* \leq K$. If the current best rule list is suboptimal, *i.e.*, $d^c \notin \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, then

$$\lambda K^* \leq \ell(d^*, \mathbf{x}, \mathbf{y}) + \lambda K^* = R(d^*, \mathbf{x}, \mathbf{y}) < R^c = R(d^c, \mathbf{x}, \mathbf{y}) = \lambda K, \tag{17}$$

in which case $K^* < K$, *i.e.*, $K^* \leq K - 1$, since K is an integer. ■

The latter part of Theorem 4 tells us that if we only need to identify a single instance of an optimal rule list $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, and we encounter a perfect K-prefix with zero misclassification error, then we can prune all prefixes of length K or greater.

Corollary 5 (Simple upper bound on prefix length) *Let $L(d)$ be the length of rule list d. For all optimal rule lists $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$,*

$$L(d^*) \leq \min\left(\left\lfloor \frac{1}{2\lambda} \right\rfloor, M\right). \tag{18}$$

Proof Let $d = ((), (), q_0, 0)$ be the empty rule list; it has objective $R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) \leq 1/2$, which gives an upper bound on R^c. Combining with (14) and Proposition 3 gives (18). ■

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

Theorem 6 (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*

$$K' < \min\left(K + \left\lfloor \frac{R^c - b(d_p, \mathbf{x}, \mathbf{y})}{\lambda} \right\rfloor, M\right). \tag{19}$$

Proof First, note that $K' \geq K$, since d'_p starts with d_p. Now recall from (12) that

$$b(d_p, \mathbf{x}, \mathbf{y}) = \ell(d, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K \leq \ell(d', \delta'_p, \mathbf{x}, \mathbf{y}) + \lambda K' = b(d'_p, \mathbf{x}, \mathbf{y}), \tag{20}$$

and from (11) that $\ell(d, \delta_p, \mathbf{x}, \mathbf{y}) \leq \ell(d', \delta'_p, \mathbf{x}, \mathbf{y})$. Combining these bounds and rearranging gives

$$b(d_p, \mathbf{x}, \mathbf{y}) + \lambda(K' - K) \leq b(d'_p, \mathbf{x}, \mathbf{y}). \tag{21}$$

Combining (21) with $b(d'_p, \mathbf{x}, \mathbf{y}) < R^c$ and Proposition 3 gives (19). ■

We can view Theorem 6 as a generalization of our one-step lookahead bound (Lemma 2), as (19) 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. Notice that when $d = ((), (), q_0, 0)$ is the empty rule list, this bound replicates (14), since $b(d_p, \mathbf{x}, \mathbf{y}) = 0$.

3.6 Upper bounds on the number of prefix evaluations

In this section, we use our upper bounds on prefix length from §3.5 to derive corresponding upper bounds on the number of prefix evaluations made by Algorithm 1. First, we present Theorem 7, 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. This number of remaining evaluations is equal to the number of prefixes that are currently in or will be inserted into the queue. 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. After Theorem 7, we present two weaker propositions that provide useful intuition.

Theorem 7 (Fine-grain upper bound on 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*

$$\Gamma(R^c, Q) \leq \sum_{d_p \in Q} \sum_{k=0}^{f(d_p)} \frac{(M - L(d_p))!}{(M - L(d_p) - k)!}, \tag{22}$$

where

$$f(d_p) = \min\left(\left\lfloor \frac{R^c - b(d_p, \mathbf{x}, \mathbf{y})}{\lambda} \right\rfloor, M - L(d_p) \right). \tag{23}$$

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 6 gives an upper bound on the length of any prefix d'_p that starts with d_p:

$$L(d'_p) \leq \min\left(L(d_p) + \left\lfloor \frac{R^c - b(d_p, \mathbf{x}, \mathbf{y})}{\lambda} \right\rfloor, M \right) \equiv U(d_p). \tag{24}$$

This gives an upper bound on the number of remaining prefix evaluations:

$$\Gamma(R^c, Q) \leq \sum_{d_p \in Q} \sum_{k=0}^{U(d_p) - L(d_p)} P(M - L(d_p), k) = \sum_{d_p \in Q} \sum_{k=0}^{f(d_p)} \frac{(M - L(d_p))!}{(M - L(d_p) - k)!}. \tag{25}$$

■

Our first 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 7, it does not use algorithm execution state to bound the size of the search space.

Proposition 8 (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,*

$$\Gamma_{\text{tot}}(S) \leq \sum_{k=0}^{K} \frac{M!}{(M - k)!}, \tag{26}$$

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

Proof By Corollary 5, $K \equiv \min(\lfloor 1/2\lambda \rfloor, M)$ gives an upper bound on the length of any optimal rule list. Since we can think of our problem as finding the optimal selection and permutation of k out of M rules, over all $k \leq K$,

$$\Gamma_{\text{tot}}(S) \leq 1 + \sum_{k=1}^{K} P(M,k) = \sum_{k=0}^{K} \frac{M!}{(M-k)!}. \tag{27}$$

■

Our next upper bound is strictly tighter than the bound in Proposition 8. Like Theorem 7, it uses the current best objective and information about the lengths of prefixes in the queue to constrain the lengths of prefixes in the remaining search space. However, Proposition 9 is weaker than Theorem 7 because it leverages only coarse-grain information from the queue. Specifically, Theorem 7 is strictly tighter because it additionally incorporates prefix-specific objective lower bound information from prefixes in the queue, which further constrains the lengths of prefixes in the remaining search space.

Proposition 9 (Coarse-grain upper bound on remaining prefix evaluations) *Consider a 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. Let Q_j be the number of prefixes of length j in Q,*

$$Q_j = \left|\{d_p : L(d_p) = j, d_p \in Q\}\right| \tag{28}$$

and let $J = \text{argmax}_{d_p \in Q} L(d_p)$ be the length of the longest prefix in Q. Define $\Gamma(R^c, Q)$ to be the number of remaining prefix evaluations, then

$$\Gamma(R^c, Q) \leq \sum_{j=1}^{J} Q_j \left(\sum_{k=0}^{K-j} \frac{(M-j)!}{(M-j-k)!} \right), \tag{29}$$

where $K = \min(\lfloor R^c/\lambda \rfloor, M)$.

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 remaining prefix d_p, Theorem 4 gives an upper bound on its length; define K to be this bound: $L(d_p) \leq \min(\lfloor R^c/\lambda \rfloor, M) \equiv K$. For any prefix d_p in queue Q with length $L(d_p) = j$, the maximum number of prefixes that start with d_p and remain to be evaluated is:

$$\sum_{k=0}^{K-j} P(M-j,k) = \sum_{k=0}^{K-j} \frac{(M-j)!}{(M-j-k)!}, \tag{30}$$

where $P(T,k)$ denotes the number of k-permutations of T. This gives an upper bound on the number of remaining prefix evaluations:

$$\Gamma(R^c, Q) \leq \sum_{j=0}^{J} Q_j \left(\sum_{k=0}^{K-j} P(M-j,k) \right) = \sum_{j=0}^{J} Q_j \left(\sum_{k=0}^{K-j} \frac{(M-j)!}{(M-j-k)!} \right). \tag{31}$$

■

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

$$\lambda < \operatorname{supp}(p_k, \mathbf{x} \,|\, d_p). \tag{32}$$

Proof Let $d^* = (d_p, \delta_p, q_0, K)$ be an optimal rule list with prefix $d_p = (p_1, \ldots, p_K)$ and labels $\delta_p = (q_1, \ldots, q_K)$. Consider the rule list $d = (d'_p, \delta'_p, q'_0, K-1)$ derived from d^* by deleting a rule $p_i \to q_i$, therefore $d'_p = (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_K)$ and $\delta'_p = (q_1, \ldots, q_{i-1}, q'_{i+1}, \ldots, q'_K)$, where q'_k need not be the same as q_k, for $k > i$ and $k = 0$.

The largest possible discrepancy between d^* and d would occur if d^* correctly classified all the data captured by p_i, while d misclassified these data. This gives an upper bound:

$$\begin{aligned} R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda(K-1) &\leq \ell(d^*, \mathbf{x}, \mathbf{y}) + \operatorname{supp}(p_i, \mathbf{x} \,|\, d_p) + \lambda(K-1) \\ &= R(d^*, \mathbf{x}, \mathbf{y}) + \operatorname{supp}(p_i, \mathbf{x} \,|\, d_p) - \lambda \\ &= R^* + \operatorname{supp}(p_i, \mathbf{x} \,|\, d_p) - \lambda \end{aligned} \tag{33}$$

where $\operatorname{supp}(p_i, \mathbf{x} \,|\, d_p)$ is the normalized support of p_i in the context of d_p, defined in (5), and the regularization 'bonus' comes from the fact that d is one rule shorter than d^*.

At the same time, we must have $R^* < R(d, \mathbf{x}, \mathbf{y})$ for d^* to be optimal. Combining this with (33) and rearranging gives (32), therefore the regularization parameter λ provides a lower bound on the support of an antecedent p_i in an optimal rule list d^*. ■

Thus, we can prune a prefix d_p if any of its antecedents do not capture more than a fraction λ of data, even if $b(d_p, \mathbf{x}, \mathbf{y}) < R^*$. Notice that the bound in Theorem 10 depends on the antecedents, but not the label predictions, and thus doesn't account for misclassification error. Theorem 11 gives a tighter bound by leveraging this additional information, which specifically tightens the upper bound on $R(d, \mathbf{x}, \mathbf{y})$ in (33).

Theorem 11 (Lower bound on accurate antecedent support) *Let d^* be any optimal rule list with objective R^*, i.e., $d^* = (d_p, \delta_p, q_0, K) \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$. Let d^* have prefix $d_p = (p_1, \ldots, p_K)$ and labels $\delta_p = (q_1, \ldots, q_K)$. For each rule $p_k \to q_k$ in d^*, define a_k to be the fraction of data that are captured by p_k and correctly classified:*

$$a_k \equiv \frac{1}{N} \sum_{n=1}^{N} \operatorname{cap}(x_n, p_k \,|\, d_p) \wedge \mathbb{1}[q_k = y_n]. \tag{34}$$

The regularization parameter λ provides a lower bound on a_k:

$$\lambda < a_k. \tag{35}$$

Proof As in Theorem 10, let $d = (d'_p, \delta'_p, q'_0, K-1)$ be the rule list derived from d^* by deleting a rule $p_i \to q_i$. Now, let us define ℓ_i to be the portion of R^* due to this rule's misclassification error,

$$\ell_i \equiv \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, p_i \,|\, d_p) \wedge \mathbb{1}[q_i \neq y_n]. \tag{36}$$

The largest discrepancy between d^* and d would occur if d misclassified all the data captured by p_i. This gives an upper bound on the difference between the misclassification error of d and d^*:

$$\begin{aligned}
\ell(d, \mathbf{x}, \mathbf{y}) - \ell(d^*, \mathbf{x}, \mathbf{y}) &\leq \text{supp}(p_i, \mathbf{x} \,|\, d_p) - \ell_i \\
&= \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, p_i \,|\, d_p) - \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, p_i \,|\, d_p) \wedge \mathbb{1}[q_i \neq y_n] \\
&= \frac{1}{N} \sum_{n=1}^{N} \text{cap}(x_n, p_i \,|\, d_p) \wedge \mathbb{1}[q_i = y_n] = a_i,
\end{aligned} \tag{37}$$

where we defined a_i in (34). Relating this bound to the objectives of d and d^* gives

$$\begin{aligned}
R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda(K-1) &\leq \ell(d^*, \mathbf{x}, \mathbf{y}) + a_i + \lambda(K-1) \\
&= R(d^*, \mathbf{x}, \mathbf{y}) + a_i - \lambda \\
&= R^* + a_i - \lambda
\end{aligned} \tag{38}$$

Combining (38) with the requirement $R^* < R(d, \mathbf{x}, \mathbf{y})$ gives the bound $\lambda < a_i$. ■

Thus, we can prune a prefix if any of its rules do not capture and correctly classify at least a fraction λ of data. While the lower bound in Theorem 10 is a sub-condition of the lower bound in Theorem 11, we can still leverage both – since the sub-condition is easier to check, checking it first can accelerate pruning. In addition to applying Theorem 10 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 greater than λ; we need not mine rules with a smaller fraction of observations. In contrast, we can only apply Theorem 11 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 Upper bound on antecedent support

In the previous section (§3.7), we proved lower bounds on antecedent support; in this section, we give an upper bound on antecedent support. Specifically, Theorem 12 shows that an antecedent's support in a rule list cannot be too similar to the set of data not captured by preceding antecedents in the rule list.

Theorem 12 (Upper bound on antecedent support) *Let $d^* = (d_p, \delta_p, q_0, K)$ be any optimal rule list with objective R^*, i.e., $d^* \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$, and let $d_p = (p_1, \ldots, p_{j-1}, p_j, \ldots, p_{K-1}, p_K)$ be its prefix. The last antecedent p_K in d_p has support*

$$\operatorname{supp}(p_K, \mathbf{x} \,|\, d_p) \leq 1 - \operatorname{supp}(d_p^{K-1}, \mathbf{x}) - \lambda, \tag{39}$$

where $d_p^{K-1} = (p_1, \ldots, p_{K-1})$, with equality implying that there also exists a shorter optimal rule list $d' = (d_p^{K-1}, \delta_p', q_0', K-1) \in \operatorname{argmin}_d R(d, \mathbf{x}, \mathbf{y})$ with prefix d_p^{K-1}. For all $k \leq K-1$, every antecedent p_k in d_p has support less than the fraction of all data not captured by preceding antecedents, by an amount greater than the regularization parameter λ:

$$\operatorname{supp}(p_k, \mathbf{x} \,|\, d_p) < 1 - \operatorname{supp}(d_p^{k-1}, \mathbf{x}) - \lambda, \tag{40}$$

where $d_p^{k-1} = (p_1, \ldots, p_{k-1})$.

Proof We begin by focusing on the last antecedent in a rule list. Let $d = (d_p, \delta_p, q_0, K)$ be a rule list with prefix $d_p = (p_1, \ldots, p_K)$ and objective $R(d, \mathbf{x}, \mathbf{y}) \leq R^*$, where $R^* \equiv \min_D R(D, \mathbf{x}, \mathbf{y})$ is the optimal objective. Also let $d' = (d_p', \delta_p', q_0', K+1)$ be a rule list whose prefix $d_p' = (p_1, \ldots, p_K, p_{K+1})$ starts with d_p and ends with a new antecedent p_{K+1}. Suppose p_{K+1} in the context of d_p' captures nearly all data not captured by d_p, except for a fraction ϵ upper bounded by the regularization parameter λ:

$$1 - \operatorname{supp}(d_p, \mathbf{x}) - \operatorname{supp}(p_{K+1}, \mathbf{x} \,|\, d_p') \equiv \epsilon \leq \lambda. \tag{41}$$

Since d_p' starts with d_p, its prefix misclassification error is at least as great; the only discrepancy between the misclassification errors of d and d' can come from the difference between the support of the set of data not captured by d_p and the support of p_{K+1}:

$$|\ell(d', \mathbf{x}, \mathbf{y}) - \ell(d, \mathbf{x}, \mathbf{y})| \leq 1 - \operatorname{supp}(d_p, \mathbf{x}) - \operatorname{supp}(p_{K+1}, \mathbf{x} \,|\, d_p') = \epsilon. \tag{42}$$

The best outcome for d' would occur if its misclassification error were smaller than that of d by ϵ, therefore

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) &= \ell(d', \mathbf{x}, \mathbf{y}) + \lambda(K+1) \\ &\geq \ell(d, \mathbf{x}, \mathbf{y}) - \epsilon + \lambda(K+1) = R(d, \mathbf{x}, \mathbf{y}) - \epsilon + \lambda \geq R(d, \mathbf{x}, \mathbf{y}) \geq R^*. \end{aligned} \tag{43}$$

d' is an optimal rule list, *i.e.*, $d' \in \operatorname{argmin}_D R(D, \mathbf{x}, y)$, if and only if $R(d', \mathbf{x}, \mathbf{y}) = R(d, \mathbf{x}, \mathbf{y}) = R^*$, which requires $\epsilon = \lambda$. Otherwise, $\epsilon < \lambda$, in which case

$$R(d', \mathbf{x}, \mathbf{y}) \geq R(d, \mathbf{x}, \mathbf{y}) - \epsilon + \lambda > R(d, \mathbf{x}, \mathbf{y}) \geq R^*, \tag{44}$$

i.e., d' is not optimal. This proves the first half of Theorem 12.

To finish, we prove the bound in (40) by contradiction. First, note that the data not captured by d_p' has normalized support $\epsilon \leq \lambda$, *i.e.*,

$$1 - \operatorname{supp}(d_p', \mathbf{x}) = 1 - \operatorname{supp}(d_p, \mathbf{x}) - \operatorname{supp}(p_{K+1}, \mathbf{x} \,|\, d_p') = \epsilon \leq \lambda. \tag{45}$$

Thus for any rule list d'' whose prefix $d_p'' = (p_1, \ldots, p_{K+1}, \ldots, p_{K'})$ starts with d_p' and ends with one or more additional rules, each additional rule p_k has support $\operatorname{supp}(p_k, \mathbf{x} \,|\, d_p'') \leq$

$\epsilon \leq \lambda$, for all $k > K+1$. By Theorem 10, all of the additional rules have insufficient support, therefore d''_p cannot be optimal, *i.e.*, $d'' \notin \operatorname{argmin}_D R(D, \mathbf{x}, \mathbf{y})$. ■

Similar to Theorem 10, our lower bound on antecedent support, we can apply Theorem 12 in the contexts of both constructing rule lists and rule mining (§3.1). Theorem 12 implies that if we only seek a single optimal rule list, then during branch-and-bound execution, we can prune a prefix if we ever add an antecedent with support too similar to the support of set of data not captured by the preceding antecedents. One way to view this result is that if $d = (d_p, \delta_p, q_0, K)$ and $d' = (d'_p, \delta'_p, q'_0, K+1)$ are rule lists such that d'_p starts with d_p and ends with an antecedent that captures all or nearly all data not captured by d_p, then the new rule in d' behaves similar to the default rule of d. As a result, the misclassification error of d' must be similar to that of d, and any reduction may not be sufficient to offset the penalty for longer prefixes. Furthermore, Theorem 12 implies that we should only mine rules with normalized support less than $1 - \lambda$; we need not mine rules with a larger fraction of observations.

3.9 Antecedent rejection and its propagation

In this section, we demonstrate further consequences of our lower (§3.7) and upper bounds (§3.8) on antecedent support, under a unified framework we refer to as antecedent rejection. Let $d_p = (p_1, \ldots, p_K)$ be a prefix, and let p_k be an antecedent in d_p. Define p_k to have insufficient support in d_p if does not obey the bound in (32) of Theorem 10. Define p_k to have insufficient accurate support in d_p if it does not obey the bound in (35) of Theorem 11. Define p_k to have excessive support in d_p if it does not obey the appropriate bound in Theorem 12, *i.e.*, either it's the last antecedent and doesn't obey (40), or it's any other antecedent and doesn't obey (39). If p_k in the context of d_p has insufficient support, insufficient accurate support, or excessive support, let us say that prefix d_p rejects antecedent p_K. Next, in Theorem 13, we describe large classes of related rule lists whose prefixes all reject the same antecedent.

Theorem 13 (Antecedent rejection propagates) *For any prefix $d_p = (p_1, \ldots, p_K)$, let $\phi(d_p)$ denote the set of all prefixes d'_p such that the set of all antecedents in d_p is a subset of the set of all antecedents in d'_p, i.e.,*

$$\phi(d_p) = \{d'_p = (p'_1, \ldots, p'_{K'}) \text{ s.t. } \{p_k : p_k \in d_p\} \subseteq \{p'_\kappa : p'_\kappa \in d'_p\}, K' \geq K\}. \tag{46}$$

Let $d = (d_p, \delta_p, q_0, K)$ be a rule list with prefix $d_p = (p_1, \ldots, p_{K-1}, p_K)$, such that d_p rejects its last antecedent p_K, either because p_K in the context of d_p has insufficient support, insufficient accurate support, or excessive support. Let $d_p^{K-1} = (p_1, \ldots, p_{K-1})$ be the first $K-1$ antecedents of d_p. Let $D = (D_p, \Delta_p, Q_0, \kappa)$ be any rule list with prefix $D_p = (P_1, \ldots, P_{K'-1}, P_{K'}, \ldots, P_\kappa)$ such that D_p starts with $D_p^{K'-1} = (P_1, \ldots, P_{K'-1}) \in \phi(d_p^{K-1})$ and antecedent $P_{K'} = p_K$. It follows that prefix D_p rejects $P_{K'}$ for the same reason that d_p rejects p_K, and furthermore, D cannot be optimal, i.e., $D \notin \operatorname{argmin}_{d^\dagger} R(d^\dagger, \mathbf{x}, \mathbf{y})$.

Proof Combine Propositions 14, 15, and 16. ■

Proposition 14 (Insufficient antecedent support propagates) *First define $\phi(d_p)$ as in (46), and let $d_p = (p_1, \ldots, p_{K-1}, p_K)$ be a prefix, such that its last antecedent p_K has insufficient support, i.e., the opposite of the bound in (32): $\text{supp}(p_K, \mathbf{x} \,|\, d_p) \leq \lambda$. Let $d_p^{K-1} = (p_1, \ldots, p_{K-1})$, and let $D = (D_p, \Delta_p, Q_0, \kappa)$ be any rule list with prefix $D_p = (P_1, \ldots, P_{K'-1}, P_{K'}, \ldots, P_\kappa)$, such that D_p starts with $D_p^{K'-1} = (P_1, \ldots, P_{K'-1}) \in \phi(d_p^{K-1})$ and $P_{K'} = p_K$. It follows that $P_{K'}$ has insufficient support in prefix D_p, and furthermore, D cannot be optimal, i.e., $D \notin \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$.*

Proof The support of p_K in d_p depends only on the set of antecedents in $d_p^K = (p_1, \ldots, p_K)$:

$$\begin{aligned}\text{supp}(p_K, \mathbf{x} \,|\, d_p) = \frac{1}{N}\sum_{n=1}^{N} \text{cap}(x_n, p_K \,|\, d_p) &= \frac{1}{N}\sum_{n=1}^{N} \left(\neg\, \text{cap}(x_n, d_p^{K-1})\right) \wedge \text{cap}(x_n, p_K) \\ &= \frac{1}{N}\sum_{n=1}^{N} \left(\bigwedge_{k=1}^{K-1} \neg\, \text{cap}(x_n, p_k)\right) \wedge \text{cap}(x_n, p_K) \leq \lambda.\end{aligned} \tag{47}$$

Similarly, the support of $P_{K'}$ in D_p depends only on the set of antecedents in $D_p^{K'} = (P_1, \ldots, P_{K'})$:

$$\begin{aligned}\text{supp}(P_{K'}, \mathbf{x} \,|\, D_p) &= \frac{1}{N}\sum_{n=1}^{N} \text{cap}(x_n, P_{K'} \,|\, D_p) \\ &= \frac{1}{N}\sum_{n=1}^{N} \left(\neg\, \text{cap}(x_n, D_p^{K'-1})\right) \wedge \text{cap}(x_n, P_{K'}) \\ &= \frac{1}{N}\sum_{n=1}^{N} \left(\bigwedge_{k=1}^{K'-1} \neg\, \text{cap}(x_n, P_k)\right) \wedge \text{cap}(x_n, P_{K'}) \\ &\leq \frac{1}{N}\sum_{n=1}^{N} \left(\bigwedge_{k=1}^{K-1} \neg\, \text{cap}(x_n, p_k)\right) \wedge \text{cap}(x_n, P_{K'}) \\ &= \frac{1}{N}\sum_{n=1}^{N} \left(\bigwedge_{k=1}^{K-1} \neg\, \text{cap}(x_n, p_k)\right) \wedge \text{cap}(x_n, p_K) = \text{supp}(p_K, \mathbf{x} \,|\, d_p) \leq \lambda.\end{aligned} \tag{48}$$

The first inequality reflects the condition that $D_p^{K'-1} \in \phi(d_p^{K-1})$, which implies that the set of antecedents in $D_p^{K'-1}$ contains the set of antecedents in d_p^{K-1}, and the next equality reflects the fact that $P_{K'} = p_K$. Thus, P'_K has insufficient support in prefix D_p, therefore by Theorem 10, D cannot be optimal, *i.e.*, $D \notin \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$. ■

Proposition 15 (Insufficient accurate antecedent support propagates) *Let $\phi(d_p)$ denote the set of all prefixes d'_p such that the set of all antecedents in d_p is a subset of the set of all antecedents in d'_p, as in (46). Let $d = (d_p, \delta_p, q_0, K)$ be a rule list with prefix*

$d_p = (p_1, \ldots, p_K)$ and labels $\delta_p = (q_1, \ldots, q_K)$, such that the last antecedent p_K has insufficient accurate support, i.e., the opposite of the bound in (35)*:*

$$\frac{1}{N}\sum_{n=1}^{N} \mathrm{cap}(x_n, p_K \,|\, d_p) \wedge \mathbb{1}[q_K = y_n] \leq \lambda. \tag{49}$$

Let $d_p^{K-1} = (p_1, \ldots, p_{K-1})$ and let $D = (D_p, \Delta_p, Q_0, \kappa)$ be any rule list with prefix $D_p = (P_1, \ldots, P_\kappa)$ and labels $\Delta_p = (Q_1, \ldots, Q_\kappa)$, such that D_p starts with $D_p^{K'-1} = (P_1, \ldots, P_{K'-1})$ $\in \phi(d_p^{K-1})$ and $P_{K'} = p_K$. It follows that $P_{K'}$ has insufficient accurate support in prefix D_p, and furthermore, $D \notin \mathrm{argmin}_{d^\dagger} R(d^\dagger, \mathbf{x}, \mathbf{y})$.

Proof The accurate support of $P_{K'}$ in D_p is insufficient:

$$\begin{aligned}
&\frac{1}{N}\sum_{n=1}^{N} \mathrm{cap}(x_n, P_{K'} \,|\, D_p) \wedge \mathbb{1}[Q_{K'} = y_n] \\
&\qquad = \frac{1}{N}\sum_{n=1}^{N} \left(\bigwedge_{k=1}^{K'-1} \neg\, \mathrm{cap}(x_n, P_k) \right) \wedge \mathrm{cap}(x_n, P_{K'}) \wedge \mathbb{1}[Q_{K'} = y_n] \\
&\qquad \leq \frac{1}{N}\sum_{n=1}^{N} \left(\bigwedge_{k=1}^{K-1} \neg\, \mathrm{cap}(x_n, p_k) \right) \wedge \mathrm{cap}(x_n, P_{K'}) \wedge \mathbb{1}[Q_{K'} = y_n] \\
&\qquad = \frac{1}{N}\sum_{n=1}^{N} \left(\bigwedge_{k=1}^{K-1} \neg\, \mathrm{cap}(x_n, p_k) \right) \wedge \mathrm{cap}(x_n, p_K) \wedge \mathbb{1}[Q_{K'} = y_n] \\
&\qquad = \frac{1}{N}\sum_{n=1}^{N} \mathrm{cap}(x_n, p_K \,|\, d_p) \wedge \mathbb{1}[Q_{K'} = y_n] \\
&\qquad \leq \frac{1}{N}\sum_{n=1}^{N} \mathrm{cap}(x_n, p_K \,|\, d_p) \wedge \mathbb{1}[q_K = y_n] \leq \lambda.
\end{aligned} \tag{50}$$

The first inequality reflects the condition that $D_p^{K'-1} \in \phi(d_p^{K-1})$, the next equality reflects the fact that $P_{K'} = p_K$. For the following equality, notice that $Q_{K'}$ is the majority class label of data captured by $P_{K'}$ in D_p, and q_K is the majority class label of data captured by P_K in d_p, and recall from (48) that $\mathrm{supp}(P_{K'}, \mathbf{x} \,|\, D_p) \leq \mathrm{supp}(p_K, \mathbf{x} \,|\, d_p)$. By Theorem 11, $D \notin \mathrm{argmin}_{d^\dagger} R(d^\dagger, \mathbf{x}, \mathbf{y})$. ■

Proposition 16 (Excessive antecedent support propagates) *Define $\phi(d_p)$ as in* (46)*, and let $d_p = (p_1, \ldots, p_K)$ be a prefix, such that its last antecedent p_K has excessive support, i.e., the opposite of the bound in* (39)*:*

$$\mathrm{supp}(p_K, \mathbf{x} \,|\, d_p) \geq 1 - \mathrm{supp}(d_p^{K-1}, \mathbf{x}) - \lambda, \tag{51}$$

where $d_p^{K-1} = (p_1, \ldots, p_{K-1})$. Let $D = (D_p, \Delta_p, Q_0, \kappa)$ be any rule list with prefix $D_p = (P_1, \ldots, P_\kappa)$ such that D_p starts with $D_p^{K'-1} = (P_1, \ldots, P_{K'-1}) \in \phi(d_p^{K-1})$ and $P_{K'} = p_K$. It follows that $P_{K'}$ has excessive support in prefix D_p, and furthermore, $D \notin \mathrm{argmin}_d R(d, \mathbf{x}, \mathbf{y})$.

Proof Since $D_p^{K'} = (P_1, \ldots, P_{K'})$ contains all the antecedents in d_p, we have that

$$\text{supp}(D_p^{K'}, \mathbf{x}) \geq \text{supp}(d_p, \mathbf{x}). \tag{52}$$

Expanding these two terms gives

$$\begin{aligned}\text{supp}(D_p^{K'}, \mathbf{x}) &= \text{supp}(D_p^{K'-1}, \mathbf{x}) + \text{supp}(P_{K'}, \mathbf{x} \,|\, D_p) \\ &\geq \text{supp}(d_p, \mathbf{x}) = \text{supp}(d_p^{K-1}, \mathbf{x}) + \text{supp}(p_K, \mathbf{x} \,|\, d_p) \geq 1 - \lambda. \end{aligned} \tag{53}$$

Rearranging gives

$$\text{supp}(P_{K'}, \mathbf{x} \,|\, D_p) \geq 1 - \text{supp}(D_p^{K'-1}, \mathbf{x}) - \lambda, \tag{54}$$

thus $P_{K'}$ has excessive support in D_p. By Theorem 12, $D \notin \text{argmin}_d R(d, \mathbf{x}, \mathbf{y})$. ■

Theorem 13 implies potentially significant computational savings. During branch-and-bound execution, if we ever encounter a prefix $d_p = (p_1, \ldots, p_{K-1}, p_K)$ that rejects its last antecedent p_K, then we can prune d_p. Furthermore, we can also prune *any* prefix d'_p whose antecedents contains the set of antecedents in d_p, in almost any order, with the constraint that all antecedent in $\{p_1, \ldots, p_{K-1}\}$ precede p_K.

3.10 Equivalent support bound

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 17 below implies that d' also has the minimum objective over all rule lists whose prefixes start with *any* prefix in $\xi(D_p)$.

Theorem 17 (Equivalent support bound) *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 $d_p = (p_1, \ldots, p_K)$, and let $D = (D_p, \Delta_p, Q_0, \kappa)$ be a rule list with prefix $D_p = (P_1, \ldots, P_\kappa)$, such that d_p and D_p capture the same data, i.e.,*

$$\{x_n : \text{cap}(x_n, d_p)\} = \{x_n : \text{cap}(x_n, D_p)\}. \tag{55}$$

If the objective lower bounds of d and D obey $b(d_p, \mathbf{x}, \mathbf{y}) \leq b(D_p, \mathbf{x}, \mathbf{y})$, 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 $\sigma(D_p)$:

$$\min_{d' \in \sigma(d_p)} R(d', \mathbf{x}, \mathbf{y}) \leq \min_{D' \in \sigma(D_p)} R(D', \mathbf{x}, \mathbf{y}). \tag{56}$$

Proof We begin by defining four related rule lists. First, let $d = (d_p, \delta_p, q_0, K)$ be a rule list with prefix $d_p = (p_1, \ldots, p_K)$ and labels $\delta_p = (q_1, \ldots, q_K)$. Second, let $D = (D_p, \Delta_p, Q_0, \kappa)$ be a rule list with prefix $D_p = (P_1, \ldots, P_\kappa)$ that captures the same data as d_p, and labels $\Delta_p = (Q_1, \ldots, Q_\kappa)$. Third, let $d' = (d'_p, \delta'_p, q'_0, K') \in \sigma(d_p)$ be any rule list whose prefix

starts with d_p, such that $K' \geq K$. Denote the prefix and labels of d' by $d'_p = (p_1, \ldots, p_K, p_{K+1}, \ldots, p_{K'})$ and $\delta_p = (q_1, \ldots, q_{K'})$, respectively. Finally, define $D' = (D'_p, \Delta'_p, Q'_0, \kappa') \in \sigma(D_p)$ to be the 'analogous' rule list, *i.e.*, whose prefix $D'_p = (P_1, \ldots, P_\kappa, P_{\kappa+1}, \ldots, P_{\kappa'}) = (P_1, \ldots, P_\kappa, p_{K+1}, \ldots, p_{K'})$ starts with D_p and ends with the same $K' - K$ antecedents as d'_p. Let $\Delta'_p = (Q_1, \ldots, Q_{\kappa'})$ denote the labels of D'.

Next, we claim that the difference in the objectives of rule lists d' and d is the same as the difference in the objectives of rule lists D' and D. Let us expand the first difference as

$$\begin{aligned} &R(d', \mathbf{x}, \mathbf{y}) - R(d, \mathbf{x}, \mathbf{y}) = \ell(d', \mathbf{x}, \mathbf{y}) + \lambda K' - \ell(d, \mathbf{x}, \mathbf{y}) - \lambda K \\ &= \ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) + \ell_0(d'_p, q'_0, \mathbf{x}, \mathbf{y}) - \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) - \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) + \lambda(K' - K). \end{aligned}$$

Similarly, let us expand the second difference as

$$\begin{aligned} &R(D', \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}) = \ell(D', \mathbf{x}, \mathbf{y}) + \lambda \kappa' - \ell(D, \mathbf{x}, \mathbf{y}) - \lambda \kappa \\ &= \ell_p(D'_p, \Delta'_p, \mathbf{x}, \mathbf{y}) + \ell_0(D'_p, Q'_0, \mathbf{x}, \mathbf{y}) - \ell_p(D_p, \Delta_p, \mathbf{x}, \mathbf{y}) - \ell_0(D_p, Q_0, \mathbf{x}, \mathbf{y}) + \lambda(K' - K), \end{aligned}$$

where we have used the fact that $\kappa' - \kappa = K' - K$.

The prefixes d_p and D_p capture the same data. Equivalently, the set of data that is not captured by d_p is the same as the set of data that is not captured by D_p, *i.e.*,

$$\{x_n : \neg \operatorname{cap}(x_n, d_p)\} = \{x_n : \neg \operatorname{cap}(x_n, D_p)\}. \tag{57}$$

Thus, the corresponding rule lists d and D share the same default rule, *i.e.*, $q_0 = Q_0$, yielding the same default rule misclassification error:

$$\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) = \ell_0(D_p, Q_0, \mathbf{x}, \mathbf{y}). \tag{58}$$

Similarly, prefixes d'_p and D'_p capture the same data, and thus rule lists d' and D' have the same default rule misclassification error:

$$\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) = \ell_0(D_p, Q_0, \mathbf{x}, \mathbf{y}). \tag{59}$$

At this point, to demonstrate our claim relating the objectives of d, d', D, and D', what remains is to show that the difference in the misclassification errors of prefixes d'_p and d_p is the same as that between D'_p and D_p. We can expand the first difference as

$$\ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) - \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n], \tag{60}$$

where we have used the fact that since d'_p starts with d_p, the first K rules in d'_p make the same mistakes as those in d_p. Similarly, we can expand the second difference as

$$\begin{aligned} \ell_p(D'_p, \Delta'_p, \mathbf{x}, \mathbf{y}) - \ell_p(D_p, \Delta_p, \mathbf{x}, \mathbf{y}) &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=\kappa+1}^{\kappa'} \operatorname{cap}(x_n, P_k \,|\, D'_p) \wedge \mathbb{1}[Q_k \neq y_n] \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, D'_p) \wedge \mathbb{1}[Q_k \neq y_n] \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] \\ &= \ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) - \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}). \end{aligned} \tag{61}$$

To justify the equality in (61), we observe first that prefixes D'_p and d'_p start with κ and K antecedents, respectively, that capture the same data. Second, prefixes D'_p and d'_p end with exactly the same ordered list of $K' - K$ antecedents, therefore for any $k = 1, \ldots, K' - K$, antecedent $P_{\kappa+k} = p_{K+k}$ in D'_p captures the same data as p_{K+k} captures in d'_p. It follows that the corresponding labels are all equivalent, *i.e.*, $Q_{\kappa+k} = q_{K+k}$, for all $k = 1, \ldots, K' - K$, and consequently, the prefix misclassification error associated with the last $K' - K$ antecedents of d'_p is the same as that of D'_p. We have therefore shown that the difference between the objectives of d' and d is the same as that between D' and D, *i.e.*,

$$R(d', \mathbf{x}, \mathbf{y}) - R(d, \mathbf{x}, \mathbf{y}) = R(D', \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}). \tag{62}$$

Next, suppose that the objective lower bounds of d and D obey $b(d_p, \mathbf{x}, \mathbf{y}) \leq b(D_p, \mathbf{x}, \mathbf{y})$, therefore

$$\begin{aligned} R(d, \mathbf{x}, \mathbf{y}) &= \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) + \lambda K \\ &= b(d_p, \mathbf{x}, \mathbf{y}) + \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) \\ &\leq b(D_p, \mathbf{x}, \mathbf{y}) + \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) = b(D_p, \mathbf{x}, \mathbf{y}) + \ell_0(D_p, Q_0, \mathbf{x}, \mathbf{y}) = R(D, \mathbf{x}, \mathbf{y}). \end{aligned} \tag{63}$$

Now let d^* be an optimal rule list with prefix constrained to start with d_p,

$$d^* \in \operatorname*{argmin}_{d^\dagger \in \sigma(d_p)} R(d^\dagger, \mathbf{x}, \mathbf{y}), \tag{64}$$

and let K^* be the length of d^*. Let D^* be the analogous κ^*-rule list whose prefix starts with D_p and ends with the same $K^* - K$ antecedents as d^*, where $\kappa^* = \kappa + K^* - K$. By (62),

$$R(d^*, \mathbf{x}, \mathbf{y}) - R(d, \mathbf{x}, \mathbf{y}) = R(D^*, \mathbf{x}, \mathbf{y}) - R(D, \mathbf{x}, \mathbf{y}). \tag{65}$$

Furthermore, we claim that D^* is an optimal rule list with prefix constrained to start with D_p,

$$D^* \in \operatorname*{argmin}_{D^\dagger \in \sigma(D_p)} R(D^\dagger, \mathbf{x}, \mathbf{y}). \tag{66}$$

To demonstrate (66), we consider two separate scenarios. In the first scenario, prefixes d_p and D_p are composed of the same antecedents, *i.e.*, the two prefixes are equivalent up to a permutation of their antecedents, and as a consequence, $\kappa = K$ and $\kappa^* = K^*$. Here, every rule list $d'' \in \sigma(d_p)$ that starts with d_p has an analogue $D'' \in \sigma(D_p)$ that starts with D_p, such that d'' and D'' obey (62), and vice versa, and thus (66) is a direct consequence of (65).

In the second scenario, prefixes d_p and D_p are not composed of the same antecedents. Define $\phi = \{p_k : (p_k \in d_p) \wedge (p_k \notin D_p)\}$ to be the set of antecedents in d_p that are not in D_p, and define $\Phi = \{P_k : (P_k \in D_p) \wedge (P_k \notin d_p)\}$ to be the set of antecedents in D_p that are not in d_p; either $\phi \neq \emptyset$, or $\Phi \neq \emptyset$, or both.

Suppose $\phi \neq \emptyset$, and let $p \in \phi$ be an antecedent in ϕ. It follows that there exists a subset of rule lists in $\sigma(D_p)$ that do not have analogues in $\sigma(d_p)$. Let $D'' \in \sigma(D_p)$ be such a rule list, such that its prefix $D''_p = (P_1, \ldots, P_\kappa, \ldots, p, \ldots)$ starts with D_p and contains p among

its remaining antecedents. Since p captures a subset of the data that d_p captures, and D_p captures the same data as d_p, it follows that p doesn't capture any data in D''_p, *i.e.*,

$$\frac{1}{N}\sum_{n=1}^{N} \text{cap}(x_n, p \,|\, D''_p) = 0 \leq \lambda. \tag{67}$$

By Theorem 10, antecedent p has insufficient support in D'', and thus D'' cannot be optimal, *i.e.*, $D'' \notin \text{argmin}_{D^\dagger \in \sigma(D_p)} R(D^\dagger, \mathbf{x}, \mathbf{y})$. By a similar argument, if $\Phi \neq \emptyset$ and $P \in \Phi$, and $d'' \in \sigma(d_p)$ is any rule list whose prefix starts with d_p and contains antecedent P, then d'' cannot be optimal, *i.e.*, $d'' \notin \text{argmin}_{d^\dagger \in \sigma(d_p)} R(d^\dagger, \mathbf{x}, \mathbf{y})$.

To finish justifying claim (66) for the second scenario, first define

$$\tau(d_p, \Phi) \equiv \{d'' = (d''_p, \delta''_p, q''_0, K'') : d'' \in \sigma(d_p) \text{ and } p_k \notin \Phi, \forall p_k \in d''_p\} \subset \sigma(d_p) \tag{68}$$

to be the set of all rule lists whose prefixes start with d_p and don't contain any antecedents in Φ. Now, recognize that the optimal prefixes in $\tau(d_p, \Phi)$ and $\sigma(d_p)$ are the same, *i.e.*,

$$\underset{d^\dagger \in \tau(d_p, \Phi)}{\text{argmin}} R(d^\dagger, \mathbf{x}, \mathbf{y}) = \underset{d^\dagger \in \sigma(d_p)}{\text{argmin}} R(d^\dagger, \mathbf{x}, \mathbf{y}), \tag{69}$$

and similarly, the optimal prefixes in $\tau(D_p, \phi)$ and $\sigma(D_p)$ are the same, *i.e.*,

$$\underset{D^\dagger \in \tau(D_p, \phi)}{\text{argmin}} R(D^\dagger, \mathbf{x}, \mathbf{y}) = \underset{D^\dagger \in \sigma(D_p)}{\text{argmin}} R(D^\dagger, \mathbf{x}, \mathbf{y}). \tag{70}$$

Since we have shown that every $d'' \in \tau(d_p, \Phi)$ has a direct analogue $D'' \in \tau(D_p, \phi)$, such that d'' and D'' obey (62), and vice versa, we again have (66) as a consequence of (65).

We can now finally combine (63) and (66) to obtain

$$\min_{d' \in \sigma(d_p)} R(d', \mathbf{x}, \mathbf{y}) = R(d^*, \mathbf{x}, \mathbf{y}) \leq R(D^*, \mathbf{x}, \mathbf{y}) = \min_{D' \in \sigma(D_p)} R(D', \mathbf{x}, \mathbf{y}). \tag{71}$$

■

Thus, if prefixes d_p and D_p capture the same data, and their objective lower bounds obey $b(d_p, \mathbf{x}, \mathbf{y}) \leq b(D_p, \mathbf{x}, \mathbf{y})$, Theorem 17 implies that we can prune D_p. Next, in Sections 3.11 and 3.12, we highlight and analyze the special case of prefixes that capture the same data because they contain the same antecedents.

3.11 Permutation bound

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. Now, let d be a rule list with prefix d_p in Π, such that d has the minimum objective over all rule lists with prefixes in Π. 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. Corollary 18 below, which can be viewed as special case of Theorem 17, implies that d' also has the minimum objective over all rule lists whose prefixes start with *any* prefix in Π.

Corollary 18 (Permutation bound) *Let π be any permutation of $\{1, \ldots, K\}$, and define $\sigma(d_p) = \{(d'_p, \delta'_p, q'_0, K') : d'_p$ starts with $d_p\}$ to be the set of all rule lists whose prefixes start with d_p. Let $d = (d_p, \delta_p, q_0, K)$ and $D = (D_p, \Delta_p, Q_0, K)$ denote rule lists with prefixes $d_p = (p_1, \ldots, p_K)$ and $D_p = (p_{\pi(1)}, \ldots, p_{\pi(K)})$, 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 $b(d_p, \mathbf{x}, \mathbf{y}) \leq b(D_p, \mathbf{x}, \mathbf{y})$, 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 $\sigma(D_p)$:*

$$\min_{d' \in \sigma(d_p)} R(d', \mathbf{x}, \mathbf{y}) \leq \min_{D' \in \sigma(D_p)} R(D', \mathbf{x}, \mathbf{y}). \tag{72}$$

Proof Since prefixes d_p and D_p contain the same antecedents, they both capture the same data. Thus, we can apply Theorem 17. ■

Thus if prefixes d_p and D_p have the same antecedents, up to a permutation, and their objective lower bounds obey $b(d_p, \mathbf{x}, \mathbf{y}) \leq b(D_p, \mathbf{x}, \mathbf{y})$, Corollary 18 implies that we can prune D_p. We call this symmetry-aware pruning, and we illustrate the subsequent computational savings next in §3.12.

3.12 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 (§3.11). 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.

First, notice that 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 19 (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,*

$$\Gamma_{\text{tot}}(S) \leq 1 + \sum_{k=1}^{K} \frac{1}{(k-1)!} \cdot \frac{M!}{(M-k)!}, \tag{73}$$

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

Proof By Corollary 5, $K \equiv \min(\lfloor 1/2\lambda \rfloor, M)$ 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 $\{1, \ldots, M\}$. 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

$$\Gamma_{\text{tot}}(S) \leq 1 + \sum_{k=1}^{K} C(M, k-1)(M-k+1) = 1 + \sum_{k=1}^{K} \frac{1}{(k-1)!} \cdot \frac{M!}{(M-k)!}. \tag{74}$$

$$\Gamma_{\text{tot}}(S) \leq 1 + \sum_{k=1}^{K} C(M, k-1)(M-k+1). \tag{75}$$

■

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 8. 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 our experiments (§6), our permutation bound in Corollary 18 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 10) we expect to obtain via rule mining (§3.1).

3.13 Similar support bound

We now present a relaxation of our equivalent support bound from Theorem 17. Note that our implementation (§5) does not currently leverage the bound in Theorem 20.

Theorem 20 (Similar support bound) *Define $\sigma(d_p)$ to be the set of all rule lists whose prefixes starts with d_p, as in (1). Let $d_p = (p_1, \ldots, p_K)$ and $D_p = (P_1, \ldots, P_\kappa)$ be prefixes that capture nearly the same data. Specifically, define ω to be the normalized support of data captured by d_p and not captured by D_p, i.e.,*

$$\omega \equiv \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, D_p) \wedge \operatorname{cap}(x_n, d_p). \tag{76}$$

Similarly, define Ω to be the normalized support of data captured by D_p and not captured by d_p, i.e.,

$$\Omega \equiv \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \operatorname{cap}(x_n, D_p). \tag{77}$$

We can bound the difference between the objectives of the optimal rule lists in $\sigma(d_p)$ and $\sigma(D_p)$ as follows:

$$\min_{D^\dagger \in \sigma(D_p)} R(D^\dagger, \mathbf{x}, \mathbf{y}) - \min_{d^\dagger \in \sigma(d_p)} R(d^\dagger, \mathbf{x}, \mathbf{y}) \geq b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) - \omega - \Omega, \tag{78}$$

where $b(d_p, \mathbf{x}, \mathbf{y})$ and $b(D_p, \mathbf{x}, \mathbf{y})$ are the objective lower bounds of d and D, respectively.

Proof We begin by defining four related rule lists. First, let $d = (d_p, \delta_p, q_0, K)$ be a rule list with prefix $d_p = (p_1, \ldots, p_K)$ and labels $\delta_p = (q_1, \ldots, q_K)$. Second, let $D = (D_p, \Delta_p, Q_0, \kappa)$ be a rule list with prefix $D_p = (P_1, \ldots, P_\kappa)$ and labels $\Delta_p = (Q_1, \ldots, Q_\kappa)$. Define ω as in (76) and Ω as in (77), and require that $\omega, \Omega \leq \lambda$. Third, let $d' = (d'_p, \delta'_p, q'_0, K') \in \sigma(d_p)$ be any rule list whose prefix starts with d_p, such that $K' \geq K$. Denote the prefix and labels of d' by $d'_p = (p_1, \ldots, p_K, p_{K+1}, \ldots, p_{K'})$ and $\delta_p = (q_1, \ldots, q_{K'})$, respectively. Finally, define $D' = (D'_p, \Delta'_p, Q'_0, \kappa') \in \sigma(D_p)$ to be the 'analogous' rule list, *i.e.*, whose prefix $D'_p = (P_1, \ldots, P_\kappa, P_{\kappa+1}, \ldots, P_{\kappa'}) = (P_1, \ldots, P_\kappa, p_{K+1}, \ldots, p_{K'})$ starts with D_p and ends with the same $K' - K$ antecedents as d'_p. Let $\Delta'_p = (Q_1, \ldots, Q_{\kappa'})$ denote the labels of D'.

The smallest possible objective for D', in relation to the objective of d', reflects both the difference between the objective lower bounds of D and d and the largest possible discrepancy between the objectives of d' and D'. The latter would occur if d' misclassified all the data corresponding to both ω and Ω while D' correctly classified this same data, thus

$$R(D', \mathbf{x}, \mathbf{y}) \geq R(d', \mathbf{x}, \mathbf{y}) + b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) - \omega - \Omega. \tag{79}$$

Now let D^* be an optimal rule list with prefix constrained to start with D_p,

$$D^* \in \operatorname*{argmin}_{D^\dagger \in \sigma(D_p)} R(D^\dagger, \mathbf{x}, \mathbf{y}), \tag{80}$$

and let κ^* be the length of D^*. Also let d^* be the analogous K^*-rule list whose prefix starts with d_p and ends with the same $\kappa^* - \kappa$ antecedents as D^*, where $K^* = K + \kappa^* - \kappa$. By (79),

$$\begin{aligned}
\min_{D^\dagger \in \sigma(D_p)} R(D^\dagger, \mathbf{x}, \mathbf{y}) &= R(D^*, \mathbf{x}, \mathbf{y}) \\
&\geq R(d^*, \mathbf{x}, \mathbf{y}) + b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) - \omega - \Omega \\
&\geq \min_{d^\dagger \in \sigma(d_p)} R(d^\dagger, \mathbf{x}, \mathbf{y}) + b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) - \omega - \Omega.
\end{aligned} \tag{81}$$

■

Theorem 20 implies that if prefixes d_p and D_p are similar, and we know the optimal objective of rule lists starting with d_p, then

$$\begin{aligned}
\min_{D' \in \sigma(D_p)} R(D', \mathbf{x}, \mathbf{y}) &\geq \min_{d' \in \sigma(d_p)} R(d', \mathbf{x}, \mathbf{y}) + b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) - \chi \\
&\geq R^c + b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) - \chi,
\end{aligned} \tag{82}$$

where R^c is the current best objective, and χ is the normalized support of the set of data captured either exclusively by d_p or exclusively by D_p. It follows that

$$\min_{D' \in \sigma(D_p)} R(D', \mathbf{x}, \mathbf{y}) \geq R^c + b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) - \chi \geq R^c \tag{83}$$

if $b(D_p, \mathbf{x}, \mathbf{y}) - b(d_p, \mathbf{x}, \mathbf{y}) \geq \chi$. To conclude, we summarize this result and combine it with our notion of lookahead from Lemma 2. During branch-and-bound execution, if we demonstrate that $\min_{d' \in \sigma(d_p)} R(d', \mathbf{x}, \mathbf{y}) \geq R^c$, then we can prune all prefixes that start with any prefix D'_p in the following set:

$$\left\{ D'_p : b(D'_p, \mathbf{x}, \mathbf{y}) + \lambda - b(d_p, \mathbf{x}, \mathbf{y}) \geq \frac{1}{N} \sum_{n=1}^{N} \mathrm{cap}(x_n, d_p) \oplus \mathrm{cap}(x_n, D'_p) \right\}. \tag{84}$$

3.14 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 datapoints to be equivalent if they are captured by exactly the same antecedents, *i.e.*, $x_i \neq x_j$ are equivalent if

$$\frac{1}{M} \sum_{m=1}^{M} \mathbb{1}[\mathrm{cap}(x_i, s_m) = \mathrm{cap}(x_j, s_m)] = 1. \tag{85}$$

Notice that we can partition a dataset into sets of equivalent points; let $\{e_u\}_{u=1}^U$ enumerate these sets. Now define $\theta(e_u)$ to be the normalized support of the minority class label with respect to set e_u, *e.g.*, let

$$e_u = \{x_n : \mathbb{1}[\mathrm{cap}(x_n, s_m) = \mathrm{cap}(x_i, s_m)]\}, \tag{86}$$

and let q_u be the minority class label among points in e_u, then

$$\theta(e_u) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u]. \tag{87}$$

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 (§6), the practical consequences can be dramatic. First, for intuition, we present a general bound in Proposition 21; next, we explicitly integrate this bound into our framework in Theorem 22.

Proposition 21 (General equivalent points bound) *Let $d = (d_p, \delta_p, q_0, K)$ be a rule list, then*

$$R(d, \mathbf{x}, \mathbf{y}) \geq \sum_{u=1}^{U} \theta(e_u) + \lambda K. \tag{88}$$

Proof Recall that the objective is $R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda K$, where the misclassification error $\ell(d, \mathbf{x}, \mathbf{y})$ is given by

$$\begin{aligned}\ell(d, \mathbf{x}, \mathbf{y}) &= \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) + \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) \\ &= \frac{1}{N} \sum_{n=1}^{N} \left(\neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[q_0 \neq y_n] + \sum_{k=1}^{K} \operatorname{cap}(x_n, p_k \,|\, d_p) \wedge \mathbb{1}[q_k \neq y_n] \right). \quad (89)\end{aligned}$$

In the context of the rule list d, each set of equivalent points is classified by either a specific antecedent p_k in d, or the default rule p_0. For a set of equivalent points u, the rule list d correctly classifies either points that have the majority class label, or points that have the minority class label. Thus, d misclassifies a number of points in u at least as great as the number of points with the minority class label. To translate this into a lower bound on $\ell(d, \mathbf{x}, \mathbf{y})$, we first sum over all sets of equivalent points, and then for each such set, count differences between class labels and the minority class label of the set, instead of counting mistakes:

$$\begin{aligned}&\ell(d, \mathbf{x}, \mathbf{y}) \\ &= \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \left(\neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[q_0 \neq y_n] + \sum_{k=1}^{K} \operatorname{cap}(x_n, p_k \,|\, d_p) \wedge \mathbb{1}[q_k \neq y_n] \right) \wedge \mathbb{1}[x_n \in e_u] \\ &\geq \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \left(\neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[y_n = q_u] + \sum_{k=1}^{K} \operatorname{cap}(x_n, p_k \,|\, d_p) \wedge \mathbb{1}[y_n = q_u] \right) \wedge \mathbb{1}[x_n \in e_u]. \quad (90)\end{aligned}$$

Next, we factor out the indicator for equivalent point set membership, which yields a term that sums to one, because every datum is either captured or not captured by prefix d_p.

$$\begin{aligned}\ell(d, \mathbf{x}, \mathbf{y}) &= \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \left(\neg \operatorname{cap}(x_n, d_p) + \sum_{k=1}^{K} \operatorname{cap}(x_n, p_k \,|\, d_p) \right) \wedge \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u] \\ &= \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} (\neg \operatorname{cap}(x_n, d_p) + \operatorname{cap}(x_n, d_p)) \wedge \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u] \\ &= \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u] = \sum_{u=1}^{U} \theta(e_u), \quad (91)\end{aligned}$$

where the final equality applies the definition of $\theta(e_u)$ in (87). Therefore, $R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda K \geq \sum_{u=1}^{U} \theta(e_u) + \lambda K$. ■

Now, recall that to obtain our lower bound $b(d_p, \mathbf{x}, \mathbf{y})$ in (10), 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 22 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 22 (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,*

$$R(d', \mathbf{x}, \mathbf{y}) \geq b(d_p, \mathbf{x}, \mathbf{y}) + b_0(d_p, \mathbf{x}, \mathbf{y}), \tag{92}$$

where

$$b_0(d_p, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u]. \tag{93}$$

Proof We derive a lower bound on the default rule misclassification error $\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y})$, analogous to the lower bound (90) on the misclassification error $\ell(d, \mathbf{x}, \mathbf{y})$ in the proof of Proposition 21. As before, we sum over all sets of equivalent points, and then for each such set, we count differences between class labels and the minority class label of the set, instead of counting mistakes made by the default rule:

$$\begin{aligned}
\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) &= \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[q_0 \neq y_n] \\
&= \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[q_0 \neq y_n] \wedge \mathbb{1}[x_n \in e_u] \\
&\geq \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[y_n = q_u] \wedge \mathbb{1}[x_n \in e_u] = b_0(d_p, \mathbf{x}, \mathbf{y}),
\end{aligned} \tag{94}$$

where the final equality comes from the definition of $b_0(d_p, \mathbf{x}, \mathbf{y})$ in (93). Since we can write the objective $R(d, \mathbf{x}, \mathbf{y})$ as the sum of the objective lower bound $b(d_p, \mathbf{x}, \mathbf{y})$ and default rule misclassification error $\ell_0(d_p, q_0, \mathbf{x}, \mathbf{y})$, applying (94) gives a lower bound on $R(d, \mathbf{x}, \mathbf{y})$:

$$\begin{aligned}
R(d, \mathbf{x}, \mathbf{y}) = \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) + \lambda K &= b(d_p, \mathbf{x}, \mathbf{y}) + \ell_0(d_p, q_0, \mathbf{x}, \mathbf{y}) \\
&\geq b(d_p, \mathbf{x}, \mathbf{y}) + b_0(d_p, \mathbf{x}, \mathbf{y}).
\end{aligned} \tag{95}$$

It follows that for any rule list $d' \in \sigma(d)$ whose prefix d'_p starts with d_p, we have

$$R(d', \mathbf{x}, \mathbf{y}) \geq b(d'_p, \mathbf{x}, \mathbf{y}) + b_0(d'_p, \mathbf{x}, \mathbf{y}). \tag{96}$$

Finally, we show that the lower bound on $R(d, \mathbf{x}, \mathbf{y})$ in (95) is not greater than the lower bound on $R(d', \mathbf{x}, \mathbf{y})$ in (96). First, let us define

$$\Upsilon(d'_p, K, \mathbf{x}, \mathbf{y}) \equiv \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u]. \tag{97}$$

Now, we write a lower bound on $b(d'_p, \mathbf{x}, \mathbf{y})$ with respect to $b(d_p, \mathbf{x}, \mathbf{y})$:

$$
\begin{aligned}
b(d'_p, \mathbf{x}, \mathbf{y}) &= \ell_p(d'_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K' = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] + \lambda K' \\
&= \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K + \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] + \lambda(K' - K) \\
&= b(d_p, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] + \lambda(K' - K) \\
&= b(d_p, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] \wedge \mathbb{1}[x_n \in e_u] + \lambda(K' - K) \\
&\geq b(d_p, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[y_n = q_u] \wedge \mathbb{1}[x_n \in e_u] + \lambda(K' - K) \\
&= b(d_p, \mathbf{x}, \mathbf{y}) + \Upsilon(d'_p, K, \mathbf{x}, \mathbf{y}) + \lambda(K' - K). \qquad (98)
\end{aligned}
$$

Next, we write $b_0(d_p, \mathbf{x}, \mathbf{y})$ with respect to $b_0(d'_p, \mathbf{x}, \mathbf{y})$,

$$
\begin{aligned}
b_0(d_p, \mathbf{x}, \mathbf{y}) &= \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u] \\
&= \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \left(\neg \operatorname{cap}(x_n, d'_p) + \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \right) \wedge \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u] \\
&= b_0(d'_p, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{u=1}^{U} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[x_n \in e_u] \wedge \mathbb{1}[y_n = q_u], \qquad (99)
\end{aligned}
$$

thus rearranging gives

$$
b_0(d'_p, \mathbf{x}, \mathbf{y}) = b_0(d_p, \mathbf{x}, y) - \Upsilon(d'_p, K, \mathbf{x}, \mathbf{y}). \qquad (100)
$$

Combining (96) with first (100) and then (98) gives:

$$
\begin{aligned}
R(d', \mathbf{x}, \mathbf{y}) &\geq b(d'_p, \mathbf{x}, \mathbf{y}) + b_0(d'_p, \mathbf{x}, \mathbf{y}) \\
&= b(d'_p, \mathbf{x}, \mathbf{y}) + b_0(d_p, \mathbf{x}, y) - \Upsilon(d'_p, K, \mathbf{x}, \mathbf{y}) \\
&\geq b(d_p, \mathbf{x}, \mathbf{y}) + \Upsilon(d'_p, K, \mathbf{x}, \mathbf{y}) + \lambda(K' - K) + b_0(d_p, \mathbf{x}, y) - \Upsilon(d'_p, K, \mathbf{x}, \mathbf{y}) \\
&= b(d_p, \mathbf{x}, \mathbf{y}) + b_0(d_p, \mathbf{x}, y) + \lambda(K' - K) \geq b(d_p, \mathbf{x}, \mathbf{y}) + b_0(d_p, \mathbf{x}, \mathbf{y}). \qquad (101)
\end{aligned}
$$

■

4. Incremental computation

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 with respect to execution time. This motivates our use of a highly optimized library, designed by Yang et al. (2016) 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. In this section, we provide explicit expressions for the incremental computations that are central to our approach. Later, in §5, we describe a cache data structure for supporting our incremental framework in practice.

For completeness, before presenting our incremental expressions, let us begin by writing down the objective lower bound and objective of the empty rule list, $d = ((), (), q_0, 0)$, the first rule list evaluated in Algorithm 1. Since its prefix contains zero rules, it has zero prefix misclassification error and also has length zero. Thus, the empty rule list's objective lower bound is zero:

$$b((), \mathbf{x}, \mathbf{y}) = \ell_p((), (), \mathbf{x}, \mathbf{y}) + \lambda \cdot 0 = 0. \tag{102}$$

Since none of the data are captured by the empty prefix, the default rule corresponds to the majority class, and the objective corresponds to the default rule misclassification error:

$$\begin{aligned} R(d, \mathbf{x}, \mathbf{y}) = \ell(d, \mathbf{x}, \mathbf{y}) + \lambda \cdot 0 &= \ell_p((), (), \mathbf{x}, \mathbf{y}) + \ell_0((), q_0, \mathbf{x}, \mathbf{y}) \\ &= b((), \mathbf{x}, \mathbf{y}) + \ell_0((), q_0, \mathbf{x}, \mathbf{y}) = \ell_0((), q_0, \mathbf{x}, \mathbf{y}). \end{aligned} \tag{103}$$

Now, we derive our incremental expressions for the objective function and its lower bound. Let $d = (d_p, \delta_p, q_0, K)$ and $d' = (d'_p, \delta'_p, q'_0, K+1)$ be rule lists such that prefix $d_p = (p_1, \ldots, p_K)$ is the parent of $d'_p = (p_1, \ldots, p_K, p_{K+1})$. Let $\delta_p = (q_1, \ldots, q_K)$ and $\delta'_p = (q_1, \ldots, q_K, q_{K+1})$ be the corresponding labels. The hierarchical structure of Algorithm 1 enforces that if we ever evaluate d', then we will have already evaluated both the objective and objective lower bound of its parent, d. We would like to reuse as much of these computations as possible in our evaluation of d'. We can write the objective lower bound of d' incrementally, with respect to the objective lower bound of d:

$$\begin{aligned} b(d'_p, \mathbf{x}, \mathbf{y}) &= \ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) + \lambda(K+1) \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K+1} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] + \lambda(K+1) \qquad (104) \\ &= \ell_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K + \lambda + \frac{1}{N} \sum_{n=1}^{N} \operatorname{cap}(x_n, p_{K+1} \,|\, d'_p) \wedge \mathbb{1}[q_{K+1} \neq y_n] \\ &= b(d_p, \mathbf{x}, \mathbf{y}) + \lambda + \frac{1}{N} \sum_{n=1}^{N} \operatorname{cap}(x_n, p_{K+1} \,|\, d'_p) \wedge \mathbb{1}[q_{K+1} \neq y_n] \\ &= b(d_p, \mathbf{x}, \mathbf{y}) + \lambda + \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge \operatorname{cap}(x_n, p_{K+1}) \wedge \mathbb{1}[q_{K+1} \neq y_n]. \qquad (105) \end{aligned}$$

Thus, if we store $b(d_p, \mathbf{x}, \mathbf{y})$, then we can reuse this quantity when computing $b(d'_p, \mathbf{x}, \mathbf{y})$. Transforming (104) into (105) yields a significantly simpler expression that is a function of the stored quantity $b(d_p, \mathbf{x}, \mathbf{y})$. For the objective of d', first let us write a naïve expression:

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) &= \ell(d', \mathbf{x}, \mathbf{y}) + \lambda(K+1) = \ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) + \ell_0(d'_p, q'_0, \mathbf{x}, \mathbf{y}) + \lambda(K+1) \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K+1} \operatorname{cap}(x_n, p_k \,|\, d'_p) \wedge \mathbb{1}[q_k \neq y_n] + \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d'_p) \wedge \mathbb{1}[q'_0 \neq y_n] + \lambda(K+1). \end{aligned} \tag{106}$$

Instead, we can compute the objective of d' incrementally with respect to its objective lower bound:

$$\begin{aligned} R(d', \mathbf{x}, \mathbf{y}) &= \ell_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) + \ell_0(d'_p, q'_0, \mathbf{x}, \mathbf{y}) + \lambda(K+1) \\ &= b(d'_p, \mathbf{x}, \mathbf{y}) + \ell_0(d'_p, q'_0, \mathbf{x}, y) \\ &= b(d'_p, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d'_p) \wedge \mathbb{1}[q'_0 \neq y_n] \\ &= b(d'_p, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{n=1}^{N} \neg \operatorname{cap}(x_n, d_p) \wedge (\neg \operatorname{cap}(x_n, p_{K+1})) \wedge \mathbb{1}[q'_0 \neq y_n]. \end{aligned} \tag{107}$$

The expression in (107) is much simpler than the naïve one in (106), and is a function of $b(d'_p, \mathbf{x}, \mathbf{y})$, which we computed in (105). Though we could compute the objective of d' incrementally with respect to that of d, doing so would in practice require that we also store $R(d, \mathbf{x}, \mathbf{y})$; we prefer the approach suggested by (107) since it avoids this additional storage overhead.

We present an incremental branch-and-bound procedure in Algorithm 2, and show the incremental computations of the objective lower bound (105) and objective (107) as two separate functions in Algorithms 3 and 4, respectively. In Algorithm 2, we use a cache to store prefixes and their objective lower bounds. Algorithm 2 additionally reorganizes the structure of Algorithm 1 to group together the computations associated with all children of a particular prefix. This has two advantages. The first is to consolidate cache queries: all children of the same parent prefix compute their objective lower bounds with respect to the parent's stored value, and we only require one cache 'find' operation for the entire group of children, instead of a separate query for each child. The second is to shrink the queue's size: instead of adding all of a prefix's children as separate queue elements, we represent the entire group of children in the queue by a single element. Since the number of children associated with each prefix is close to the total number of possible antecedents, both of these effects can yield significant savings. For example, if we are trying to optimize over rule lists formed from a set of 1000 antecedents, then the maximum queue size in Algorithm 2 will be smaller than that in Algorithm 1 by a factor of nearly 1000.

5. Implementation

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

Algorithm 2 Incremental branch-and-bound for learning rule lists, for simplicity, from a cold start. We explicitly show the incremental objective lower bound and objective functions in Algorithms 3 and 4, respectively.

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 $(\mathbf{x}, \mathbf{y}) = \{(x_n, y_n)\}_{n=1}^{N}$, regularization parameter λ
Output: Provably optimal rule list d^* with minimum objective R^*

$d^c \leftarrow ((), (), q_0, 0)$ ▷ Initialize current best rule list with empty rule list
$R^c \leftarrow R(d^c, \mathbf{x}, \mathbf{y})$ ▷ Initialize current best objective
$Q \leftarrow \text{queue}([\,()\,])$ ▷ Initialize queue with empty prefix
$C \leftarrow \text{cache}([\,(()\,, 0)\,])$ ▷ Initialize cache with empty prefix and its objective lower bound
while Q not empty **do** ▷ Optimization complete when the queue is empty
  $d_p \leftarrow Q.\text{pop}()$ ▷ Remove a prefix d_p from the queue
  $b(d_p, \mathbf{x}, \mathbf{y}) \leftarrow C.\text{find}(d_p)$ ▷ Look up d_p's lower bound in the cache
  $\mathbf{u} \leftarrow \neg\,\text{cap}(\mathbf{x}, d_p)$ ▷ Bit vector indicating data not captured by d_p
  for s in S **do** ▷ Evaluate all of d_p's children
    if s not in d_p **then**
      $D_p \leftarrow (d_p, s)$ ▷ **Branch**: Generate child D_p
      $\mathbf{v} \leftarrow \mathbf{u} \wedge \text{cap}(\mathbf{x}, s)$ ▷ Bit vector indicating data captured by s in D_p
      $b(D_p, \mathbf{x}, \mathbf{y}) \leftarrow b(d_p, \mathbf{x}, \mathbf{y}) + \lambda +$ IncrementalLowerBound$(\mathbf{v}, \mathbf{y}, N)$
      if $b(D_p, \mathbf{x}, \mathbf{y}) < R^c$ **then** ▷ **Bound**: Apply bound from Theorem 1
        $R(D, \mathbf{x}, \mathbf{y}) \leftarrow b(D_p, \mathbf{x}, \mathbf{y}) +$ IncrementalObjective$(\mathbf{u}, \mathbf{v}, \mathbf{y}, N)$
        if $R(D, \mathbf{x}, \mathbf{y}) < R^c$ **then**
          $(d^c, R^c) \leftarrow (D, R(D, \mathbf{x}, \mathbf{y}))$ ▷ Update current best rule list and objective
        end if
        $Q.\text{push}(D_p)$ ▷ Add D_p to the queue
        $C.\text{insert}(D_p, b(D_p, \mathbf{x}, \mathbf{y}))$ ▷ Add D_p and its lower bound to the cache
      end if
    end if
  end for
end while
$(d^*, R^*) \leftarrow (d^c, R^c)$ ▷ Identify provably optimal rule list and objective

rule lists we have already evaluated. 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. The symmetry-aware map dominates memory usage for problems that explore longer prefixes. Before inserting per-

Algorithm 3 Incremental objective lower bound (105) used in Algorithm 2.

Input: Bit vector $\mathbf{v} \in \{0,1\}^N$ indicating data captured by s, the last antecedent in D_p, bit vector of class labels $\mathbf{y} \in \{0,1\}^N$, number of observations N
Output: Component of D's misclassification error due to data captured by s

function IncrementalLowerBound($\mathbf{v}, \mathbf{y}, N$)
  $n_v = \text{sum}(\mathbf{v})$ ▷ Number of data captured by s, the last antecedent in D_p
  $\mathbf{w} \leftarrow \mathbf{v} \wedge \mathbf{y}$ ▷ Bit vector indicating data captured by s with label 1
  $n_w = \text{sum}(\mathbf{w})$ ▷ Number of data captured by s with label 1
  if $n_w/n_v > 0.5$ **then**
    return $(n_v - n_w)/N$ ▷ Misclassification error of the rule $s \to 1$
  else
    return n_w/N ▷ Misclassification error of the rule $s \to 0$
  end if
end function

Algorithm 4 Incremental objective function (107) used in Algorithm 2.

Input: Bit vector $\mathbf{u} \in \{0,1\}^N$ indicating data not captured by D_p's parent prefix, bit vector $\mathbf{v} \in \{0,1\}^N$ indicating data not captured by s, the last antecedent in D_p, bit vector of class labels $\mathbf{y} \in \{0,1\}^N$, number of observations N
Output: Component of D's misclassification error due to its default rule

function IncrementalObjective($\mathbf{u}, \mathbf{v}, \mathbf{y}, N$)
  $\mathbf{f} \leftarrow \mathbf{u} \wedge \neg\, \mathbf{v}$ ▷ Bit vector indicating data not captured by D_p
  $n_f = \text{sum}(\mathbf{f})$ ▷ Number of data not captured by D_p
  $\mathbf{g} \leftarrow \mathbf{f} \wedge \mathbf{y}$ ▷ Bit vector indicating data not captured by D_p with label 1
  $n_g = \text{sum}(\mathbf{w})$ ▷ Number of data not captued by D_p with label 1
  if $n_f/n_g > 0.5$ **then**
    return $(n_f - n_g)/N$ ▷ Default rule misclassification error with label 1
  else
    return n_g/N ▷ Default rule misclassification error with label 0
  end if
end function

mutation P_i into the symmetry-aware map, we check if there exists a permutation P_j of P_i already in the map. If 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. Otherwise 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. A first-in-first-out (FIFO) queue implements breadth-first search (BFS), and a priority queue implements best-first search. Example priority queue policies include ordering by the lower bound, the objective, or a custom metric that maps prefixes to real values. We also support a stochastic exploration

process that bypasses the need for a queue by instead following random paths from the root to leaves. We find that ordering by the lower bound and other priority metrics often leads to a shorter runtime than using BFS.

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 small number of times.

Our implementation of CORELS is at `https://github.com/nlarusstone/corels`.

6. Experiments

Our experimental analysis addresses five questions: (1) How does CORELS' accuracy compare to other algorithms? (2) How does CORELS' model size compare to other algorithms? (3) How rapidly does the objective function converge? (4) How rapidly does CORELS prune the search space? (5) How much does each of the implementation optimizations contribute to CORELS' performance?

All results that we present were executed on a server with two Intel Xeon E5-2699 v4 (55 MB cache, 2.20 GHz) processors and 448 GB RAM. Except where we mention a memory constraint, all experiments can run comfortably on smaller machines, *e.g.*, a laptop with 16GB RAM.

Our evaluation focuses on two socially-important prediction problems associated with recent, publicly-available datasets:

- Predicting which individuals in the ProPublica COMPAS dataset (Larson et al., 2016) recidivate within two years.
- Using the NYCLU 2014 stop-and-frisk dataset (New York Civil Liberties Union, 2014) to predict whether a weapon will be found on a stopped individual who is frisked or searched.

Our choice of and approach to the second problem is inspired by the work of Goel et al. (2016), who develop regression models to analyze racial disparities in New York City's stop-and-frisk policy, for a similar, larger dataset. In particular, the authors arrive at a simple

if (*location* = *transit authority*) **then predict** *yes*
else if (*stop reason* = *suspicious object*) **then predict** *yes*
else if (*stop reason* = *suspicious bulge*) **then predict** *yes*
else predict *no*

Figure 3: An example rule list that predicts whether a weapon will be found on a stopped individual who is frisked or searched, for the NYCLU stop-and-frisk dataset. This is the most common optimal rule list found by CORELS across 10 cross-validation folds; the others contain the same rules, up to a permutation.

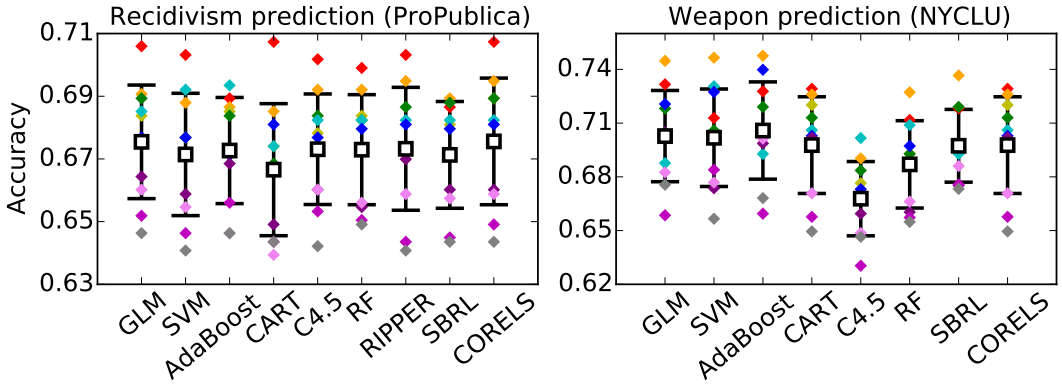

Figure 4: Comparison of CORELS and a panel of eight other algorithms: logistic regression (GLM), support vector machines (SVM), AdaBoost, CART, C4.5, random forests (RF), RIPPER, scalable Bayesian rule lists (SBRL). Test accuracy means (white squares), standard deviations (error bars), and values (colors correspond to folds), for 10-fold cross-validation experiments. Left: Two-year recidivism prediction for the ProPublica COMPAS dataset. For CORELS, we use regularization parameter $\lambda = 0.005$. Right: Weapon prediction for the NYCLU stop-and-frisk dataset. For CORELS, we use $\lambda = 0.01$. Note that we were unable to execute RIPPER for the NYCLU problem.

and interpretable heuristic that could potentially help police officers more effectively decide when to frisk and/or search stopped individuals, *i.e.*, when such interventions are likely to discover criminal possession of a weapon.

We first ran a 10-fold cross validation experiment using CORELS and eight other algorithms: logistic regression, support vector machines, AdaBoost, CART, C4.5, random forests, RIPPER, and scalable Bayesian rule lists (SBRL). [3] We use standard R packages, with default parameter settings, for the first seven algorithms. [4]

Figures 1 and 3 show example optimal rule lists that CORELS learns for the ProPublica and NYCLU datasets, respectively. While our goal is to provide illustrative examples, and not to provide a detailed analysis nor to advocate for the use of these specific models, we note that these rule lists are short and easy to understand. In particular, the three-rule list for weapon prediction in Figure 3 has the spirit of the heuristic strategy presented by Goel

3. For SBRL, we use the C implementation at `https://github.com/Hongyuy/sbrlmod`.
4. For CART, C4.5 (J48), and RIPPER, *i.e.*, the tree and rule list learning algorithms, we use the implementations from the R packages rpart, RWeka, and caret, respectively. By default, CART uses complexity parameter $cp = 0.01$, and C4.5 uses complexity parameter $C = 0.25$.

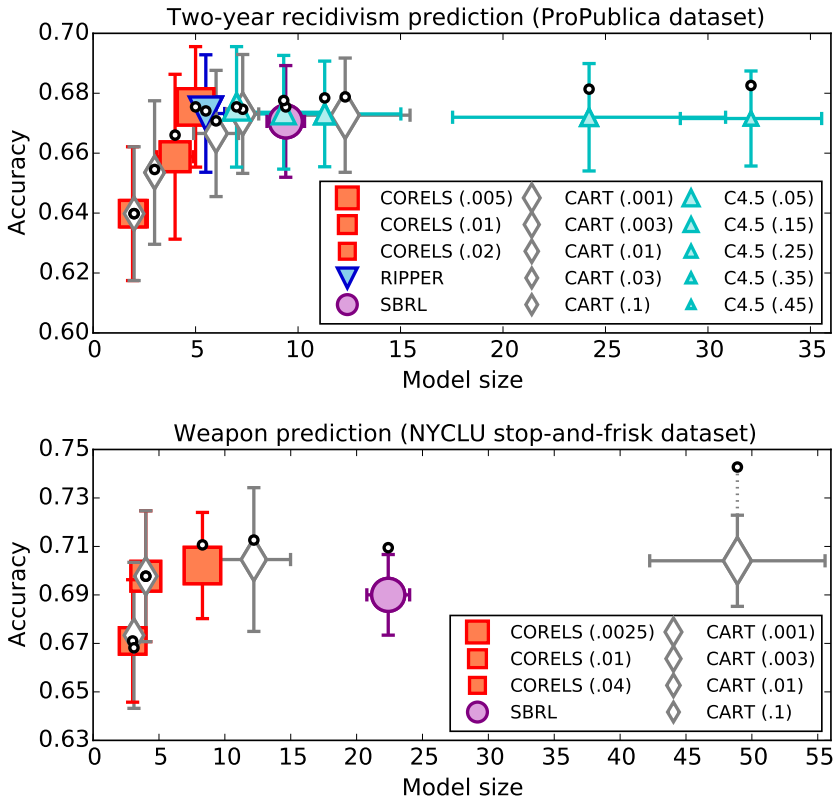

Figure 5: Training and test accuracy as a function of model size. For CORELS, CART, and C4.5, we vary the regularization parameter λ, and complexity parameters cp and C, respectively; numbers within parentheses in the legend indicate parameter values. Note that the CART implementation sets $cp = 0.01$ by default, and C4.5 uses $C = 0.25$. Legend markers and error bars indicate means and standard deviations, respectively, of test accuracy across cross-validation folds. Small circles mark associated training accuracy means. Top: Two-year recidivism prediction for the ProPublica COMPAS dataset. None of the models exhibit significant overfitting: mean training accuracy never exceeds mean test accuracy by more than about 0.01. Bottom: Weapon prediction for the NYCLU stop-and-frisk dataset. Only CART with $cp = 0.001$ significantly overfits. We do not depict C4.5, which finds large models (> 100 leaves) and dramatically overfits for all tested parameters.

et al. (2016) that combines three stop criteria and is based on a reduced version of their full regression model.

Figure 4 shows that there were no statistically significant differences in algorithm accuracies. In fact, 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 5 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.

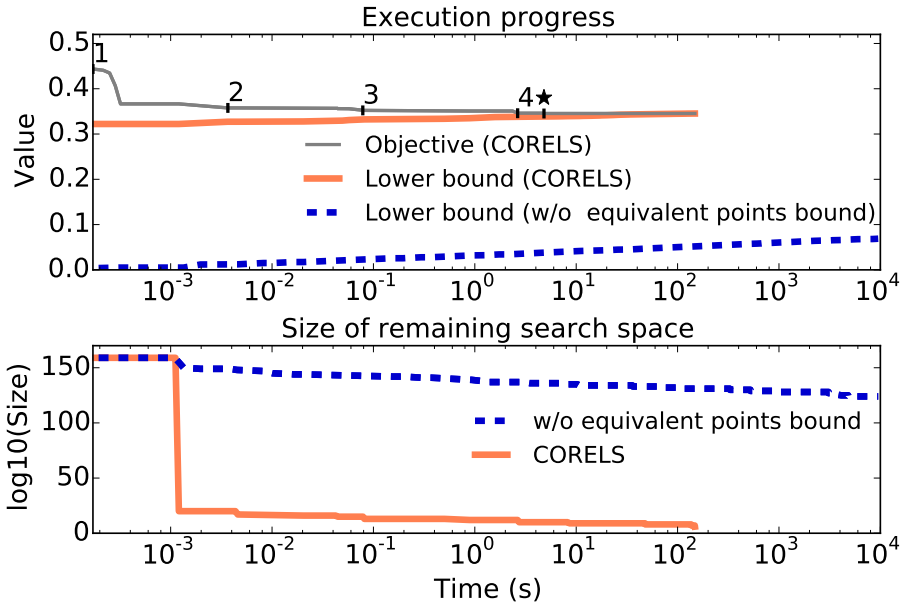

Figure 6: CORELS with (lines) and without (dashes) the equivalent points bound (Theorem 22). Top: Objective value (thin line) and lower bound (thick line) for CORELS, as a function of wall clock time (log scale). Numbered hatch marks along the trace of the objective value indicate when the length of the best known rule list changes, and are labeled by the new length. CORELS quickly achieves the optimal value (star marker), and certifies optimality when the lower bound matches the objective value. A separate execution of CORELS without the equivalent points bound remains far from complete, and its lower bound (dashed line) far from the optimum. Bottom: $\lfloor \log_{10} \Gamma(R^c, Q) \rfloor$, as a function of wall clock time (log scale), where $\Gamma(R^c, Q)$ is the upper bound on remaining search space size (Theorem 7).

In the remainder, we show results using the ProPublica dataset. The solid lines in Figure 6 illustrate how both the objective (top) and the size of the remaining search space (bottom) decrease as CORELS executes. The objective drops quickly, achieving the optimal value within 10 seconds. CORELS certifies optimality in less than 6 minutes – the objective lower bound of the remaining search space steadily converges to the optimal objective as the search space shrinks.

Finally, we determine the efficacy of each of our bounds and data structure optimizations. Both panels of Figure 6 also highlight a separate execution of CORELS without the equivalent points bound. After nearly 3 hours, the execution is still far from complete; in particular, the lower bound is far from the optimum objective value. Table 1 provides summary statistics for experiments using the full CORELS implementation and five variants that each remove a specific optimization. Figure 7 presents a view of the same experiments. These plots depict the number of prefixes of a given length in the queue during the algorithm's execution.

Removed component	t_{total} (min)	t_{opt} (s)	i_{total} ($\times 10^6$)	$Q_{\max}$ ($\times 10^6$)	$K_{\max}$
none (CORELS)	5.5 (1.6)	8 (2)	1.7 (0.4)	1.3 (0.4)	5-6
priority queue (BFS)	6.7 (2.2)	4 (1)	1.9 (0.6)	1.5 (0.5)	5-6
support bounds	10.2 (3.4)	13 (4)	2.7 (0.8)	2.2 (0.7)	5-6
symmetry-aware map	58.6 (23.3)	23 (6)	16.0 (5.9)	14.5 (5.7)	5-6
lookahead bound	71.9 (23.0)	9 (2)	18.5 (5.9)	16.3 (5.3)	6-7
equivalent pts bound	>134	>7168*	>800	>789	≥10

Table 1: Per-component performance improvement. The 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 a specific implementation optimization or bound. (We are not measuring the cumulative effects of removing a sequence of components.) All rows represent complete executions, except for the final row, in which each execution was terminated due to memory constraints, once the size of the cache reached 8×10^8 elements, after consuming 390-410GB 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.
* Only 4 out of 10 folds achieve the optimum before being terminated.

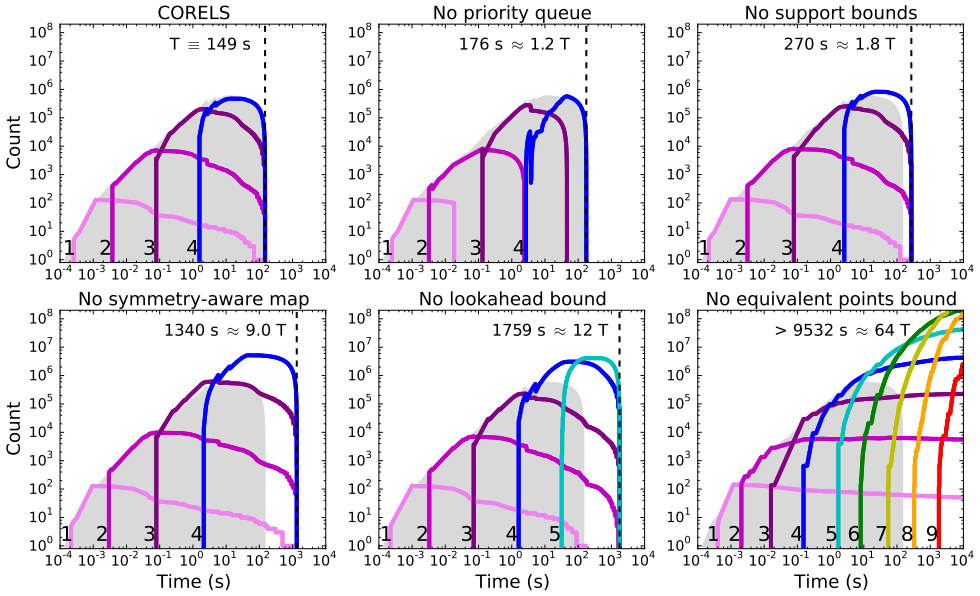

Figure 7: Logical queue composition. Numbers of prefixes in the queue (log scale), labeled and colored by length, as a function of wall clock time (log scale), for full CORELS (top left), and five variants that each remove a specific implementation optimization or bound. The gray shading fills in the 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 the other three subfigures. For each execution, we indicate the total time both in seconds, and relative to the full CORELS implementation (T = 149 s).

7. 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.

Acknowledgments

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. E.A. would like to thank E. Jonas, E. Kohler, and S. Tu for early implementation guidance, A. D'Amour for pointing out the work by Goel et al. (2016), J. Schleier-Smith and E. Thewalt for helpful conversations, and members of RISELab, SAIL, and the UC Berkeley Database Group for their support and feedback. We thank H. Yang and B. Letham for sharing advice and code for processing data and mining rules.

References

K. P. Bennett and J. A. Blue. Optimal decision trees. Technical report, R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute, 1996.

I. Bratko. Machine learning: Between accuracy and interpretability. In *Learning, Networks and Statistics*, volume 382 of *International Centre for Mechanical Sciences*, pages 163–177. Springer Vienna, 1997. ISBN 978-3-211-82910-3.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

C. Chen and C. Rudin. Optimized falling rule lists and softly falling rule lists. Work in progress, 2017.

H. A. Chipman, E. I. George, and R. E. McCulloch. Bayesian CART model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.

H. A. Chipman, E. I. George, and R. E. McCulloch. Bayesian treed models. *Machine Learning*, 48(1/3):299–320, 2002.

H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010.

W. W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning (ICML)*, pages 115–123, 1995.

R. M. Dawes. The robust beauty of improper linear models in decision making. *American Psychologist*, 34(7):571–582, 1979.

D. Dension, B. Mallick, and A.F.M. Smith. A Bayesian CART algorithm. *Biometrika*, 85 (2):363–377, 1998.

D. Dobkin, T. Fulton, D. Gunopulos, S. Kasif, and S. Salzberg. Induction of shallow decision trees, 1996.

A. Farhangfar, R. Greiner, and M. Zinkevich. A fast way to produce optimal fixed-depth decision trees. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2008)*, 2008.

A. A. Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10, 2014.

M. Garofalakis, D. Hyun, R. Rastogi, and K. Shim. 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)*, pages 335–339, 2000.

C. Giraud-Carrier. Beyond predictive accuracy: What? In *Proceedings of the ECML-98 Workshop on Upgrading Learning to Meta-Level: Model Selection and Data Transformation*, pages 78–85, 1998.

S. Goel, J. M. Rao, and R. Shroff. Precinct or prejudice? Understanding racial disparities in New York City's stop-and-frisk policy. *Ann. Appl. Stat.*, 10(1):365–394, 03 2016.

M. Goessling and S. Kang. Directional decision lists. *Preprint at arXiv:1508.07643*, Aug 2015.

B. Goodman and S. Flaxman. EU regulations on algorithmic decision-making and a "right to explanation". *Preprint at arXiv:1606.08813*, 2016.

R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–91, 1993.

J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

H. Lakkaraju and C. Rudin. Cost-sensitive and interpretable dynamic treatment regimes based on rule lists. In *Proceedings of the Artificial Intelligence and Statistics (AISTATS)*, 2017.

J. Larson, S. Mattu, L. Kirchner, and J. Angwin. How we analyzed the COMPAS recidivism algorithm. *ProPublica*, 2016.

N. L. Larus-Stone. *Learning Certifiably Optimal Rule Lists: A Case For Discrete Optimization in the 21st Century.* 2017. Undergraduate thesis, Harvard College.

B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics*, 9(3):1350–1371, 2015.

W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. *IEEE International Conference on Data Mining*, pages 369–376, 2001.

B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, KDD '98, pages 80–96, 1998.

M. Marchand and M. Sokolova. Learning with decision lists of data-dependent features. *Journal of Machine Learning Research*, 6:427–451, 2005.

T. H. McCormick, C. Rudin, and D. Madigan. Bayesian hierarchical rule modeling for predicting medical conditions. *The Annals of Applied Statistics*, 6:652–668, 2012.

S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

New York Civil Liberties Union. Stop-and-frisk data, 2014. `http://www.nyclu.org/content/stop-and-frisk-data`.

S. Nijssen and E. Fromont. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1):9–51, 2010. ISSN 1384-5810.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, November 1987.

C. Rudin and S. Ertekin. Learning optimized lists of rules with mathematical programming. Unpublished, 2015.

C. Rudin, B. Letham, and D. Madigan. Learning theory analysis for association rules and sequential event prediction. *Journal of Machine Learning Research*, 14:3384–3436, 2013.

S. Rüping. *Learning interpretable models*. PhD thesis, Universität Dortmund, 2006.

G. Shmueli. To explain or to predict? *Statistical Science*, 25(3):289–310, August 2010. ISSN 0883-4237.

M. Sokolova, M. Marchand, N. Japkowicz, and J. Shawe-Taylor. The decision list machine. In *Advances in Neural Information Processing Systems*, volume 15 of *NIPS '03*, pages 921–928, 2003.

K. Vanhoof and B. Depaire. Structure of association rule classifiers: A review. In *Proceedings of the International Conference on Intelligent Systems and Knowledge Engineering*, ISKE '10, pages 9–12, 2010.

A. Vellido, J. D. Martín-Guerrero, and P. J.G. Lisboa. Making machine learning models interpretable. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2012.

F. Wang and C. Rudin. Falling rule lists. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, 2015.

H. Yang, C. Rudin, and M. Seltzer. Scalable Bayesian rule lists. *Preprint at arXiv:1602.08610*, 2016.

X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, ICDM '03, pages 331–335, 2003.

Y. Zhang, E. B. Laber, A. Tsiatis, and M. Davidian. Using decision lists to construct interpretable and parsimonious treatment regimes. *Preprint at arXiv:1504.07715*, April 2015.