Scalable Bayesian Rule Lists

Hongyu Yang HONGYUY@MIT.EDU
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, USA

Cynthia Rudin RUDIN@MIT.EDU
Computer Science and Artificial Intelligence Laboratory, and
Sloan School of Management
Massachusetts Institute of Technology, USA

Margo Seltzer MARGO@EECS.HARVARD.EDU
School of Engineering and Applied Sciences
Harvard University, USA

Abstract

We present an algorithm for building rule lists that is two orders of magnitude faster than previous work. Rule list algorithms are competitors for decision tree algorithms. They are associative classifiers, in that they are built from pre-mined association rules. They have a logical structure that is a sequence of IF-THEN rules, identical to a decision list or one-sided decision tree. Instead of using greedy splitting and pruning like decision tree algorithms, we fully optimize over rule lists, striking a practical balance between accuracy, interpretability, and computational speed. The algorithm presented here uses a mixture of theoretical bounds (tight enough to have practical implications as a screening or bounding procedure), computational reuse, and highly tuned language libraries to achieve computational efficiency. Currently, for many practical problems, this method achieves better accuracy and sparsity than decision trees; further, in many cases, the computational time is practical and often less than that of decision trees.

1. Introduction

Our goal is to build a competitor for decision tree algorithms in terms of accuracy, interpretability, and computational speed. Decision trees are widely used, particularly in industry, because of their interpretability. Their logical IF-THEN structure allows predictions to be explained to users. However, decision tree algorithms have the serious flaw that they are constructed using greedy splitting from the top down. They also use greedy pruning of nodes. They do not globally optimize any function, instead they are comprised entirely of local optimization heuristics. If the algorithm makes a mistake in the splitting near the top of the tree, it is difficult to undo it, and consequently the trees become long and uninterpretable, unless they are heavily pruned, in which case accuracy suffers. In general, decision tree algorithms are computationally tractable, not particularly accurate, and less sparse and interpretable than they could be. This leaves users with no good alternative if they desire an accurate yet sparse logical classifier.

Several important ingredients provide the underpinning for our method including:

(i) A principled objective, which is the posterior distribution for the Bayesian Rule List (BRL) model of Letham et al.'s (2015). We optimize this objective over rule lists. Our algorithm is called Scalable Bayesian Rule Lists (SBRL).

(ii) A useful statistical approximation that narrows the search space. We assume that each leaf of the rule list contains ("captures") a number of observations that is bounded below by zero. Because of this approximation, the set of conditions defining each leaf is a frequent pattern. This means the rule list is comprised of only frequent patterns. All of the possible frequent patterns can be pre-mined from the dataset using one of the standard frequent pattern mining methods. This leaves us with a much smaller optimization problem: we optimize over the set of possible pre-mined rules and their order to create the rule list.

(iii) High performance language libraries to achieve computational efficiency. Optimization over rule lists can be solved by repeated low level computations that have the capacity to be sped up. At every iteration, we make a change to the rule list and need to evaluate the new rule list on the data. The high performance calculations speed up this evaluation.

(iv) Computational reuse. When we evaluate a rule list on the data that has been modified from a previous rule list, we need only to change the evaluation of points below the change in the rule list. Thus we can reuse the computation above the change.

(v) Analytical bounds on BRL's posterior that are tight enough to be used in practice for screening association rules and providing bounds on the optimal solution. These are provided in two theorems in this paper.

Through a series of controlled experiments, our experiments show over two orders of magnitude speedup over the previous best code for this problem.

Let us provide some sample results. Figure 1 presents an example of a rule list that we learned for the UCI Mushroom dataset (see Bache & Lichman, 2013). This rule list is a predictive model for whether a mushroom is poisonous. It was created in about 10 seconds on a laptop and achieves perfect out-of-sample accuracy. Figure 2 presents a rule list for the UCI Adult dataset (see Bache & Lichman, 2013). We ran our SBRL algorithm for approximately 45 seconds on a laptop to produce this. The algorithm achieves a higher out-of-sample AUC (area under the ROC Curve) than that achieved if CART or C4.5 were heavily tuned on the test set itself.

2. Previous Work: Review of Bayesian Rule Lists of Letham et al. (2015)

Scalable Rule Lists uses the posterior distribution of the Bayesian Rule Lists algorithm. Our training set is $\{(x_i, y_i)\}_{i=1}^n$ where the $x_i \in \mathcal{X}$ encode features, and y_i are labels, which in our case are binary, either 0 or 1. A Bayesian decision list has the following form:

if x obeys a_1 **then** $y \sim \text{Binomial}(\theta_1)$, $\theta_1 \sim \text{Beta}(\boldsymbol{\alpha} + \mathbf{N}_1)$
else if x obeys a_2 **then** $y \sim \text{Binomial}(\theta_2)$, $\theta_2 \sim \text{Beta}(\boldsymbol{\alpha} + \mathbf{N}_2)$
$\vdots$

if	bruises=no,odor=not-in-(none,foul)	then probability that the mushroom is edible	= 0.00112
else if	odor=foul,gill-attachment=free,	then probability that the mushroom is edible	= 0.0007
else if	gill-size=broad,ring-number=one,	then probability that the mushroom is edible	= 0.999
else if	stalk-root=unknown,stalk-surface-above-ring=smooth,	then probability that the mushroom is edible	= 0.996
else if	stalk-root=unknown,ring-number=one,	then probability that the mushroom is edible	= 0.0385
else if	bruises=foul,veil-color=white,	then probability that the mushroom is edible	= 0.995
else if	stalk-shape=tapering,ring-number=one,	then probability that the mushroom is edible	= 0.986
else if	habitat=paths,	then probability that the mushroom is edible	= 0.958
else	(default rule)	then probability that the mushroom is edible	= 0.001

Figure 1: Rule list for the mushroom dataset from the UCI repository (data available from Bache & Lichman, 2013).

if	capital-gain>$7298.00	then probability to make over 50K	= 0.986
else if	Young,Never-married,	then probability to make over 50K	= 0.003
else if	Grad-school,Married,	then probability to make over 50K	= 0.748
else if	Young,capital-loss=0,	then probability to make over 50K	= 0.072
else if	Own-child,Never-married,	then probability to make over 50K	= 0.015
else if	Bachelors,Married,	then probability to make over 50K	= 0.655
else if	Bachelors,Over-time,	then probability to make over 50K	= 0.255
else if	Exec-managerial,Married,	then probability to make over 50K	= 0.531
else if	Married,HS-grad,	then probability to make over 50K	= 0.300
else if	Grad-school,	then probability to make over 50K	= 0.266
else if	Some-college,Married,	then probability to make over 50K	= 0.410
else if	Prof-specialty,Married,	then probability to make over 50K	= 0.713
else if	Assoc-degree,Married,	then probability to make over 50K	= 0.420
else if	Part-time,	then probability to make over 50K	= 0.013
else if	Husband,	then probability to make over 50K	= 0.126
else if	Prof-specialty,	then probability to make over 50K	= 0.148
else if	Exec-managerial,Male,	then probability to make over 50K	= 0.193
else if	Full-time,Private,	then probability to make over 50K	= 0.026
else	(default rule)	then probability to make over 50K	= 0.066.

Figure 2: Rule list for the Adult dataset from the UCI repository (see Bache & Lichman, 2013).

else if x obeys a_m **then** $y \sim \text{Binomial}(\theta_m)$, $\theta_m \sim \text{Beta}(\boldsymbol{\alpha} + \mathbf{N}_m)$
else $y \sim \text{Binomial}(\theta_0)$, $\theta_0 \sim \text{Beta}(\boldsymbol{\alpha} + \mathbf{N}_0)$.

Here, the antecedents $\{a_j\}_{j=1}^m$ are conditions on the x's that are either true or false, for instance, if x is a patient, a_j is true when x's age is above 60 years old and x has diabetes, otherwise false. The vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_0]$ has a prior parameter for each of the two labels. Values α_1 and α_0 are prior parameters, in the sense that each rule's prediction $y \sim \text{Binomial}(\theta_j)$, and $\theta_j | \boldsymbol{\alpha} \sim \text{Beta}(\boldsymbol{\alpha})$. The notation $\mathbf{N}_j$ is the vector of counts, where $N_{j,l}$ is the number of observations x_i that satisfy condition a_j but none of the previous conditions $a_1, ..., a_{j-1}$, and that have label $y_i = l$, where l is either 1 or 0. $\mathbf{N}_j$ is added to the prior parameters $\boldsymbol{\alpha}$ from the usual derivation of the posterior for the Beta-binomial. The default rule is at the bottom, which makes predictions for observations that are not satisfied by any of the conditions. When an observation satisfies condition a_j but not $a_1, ..., a_{j-1}$ we say that the observation is *captured* by rule j. Formally:

Definition 1 *Rule j* ***captures*** *observation i, denoted Captr$(i) = j$, when*

$$j = \textit{argmin } j' \textit{ such that } a_{j'}(x_i) = \textit{True}.$$

Bayesian Rule Lists is an associative classification method, in the sense that the antecedents are first mined from the database, and then the set of rules and their order are learned. The rule mining step is fast, and there are fast parallel implementations available. Any frequent pattern mining method will suffice, since the method needs only to produce all conditions with sufficiently high support in the database. The support of antecedent a_j is denoted supp(a_j), which is the number of observations that obey condition a_j. A condition is a conjunction of expressions "feature∈values," e.g., age∈[40-50] and color=white. The hard part is learning the rule list, which is what this paper focuses on.

The likelihood for the model discussed above is:

$$\text{Likelihood} = p(\mathbf{y}|\mathbf{x}, d, \alpha) \propto \prod_{j=0}^{m} \frac{\Gamma(N_{j,0} + \alpha_0)\Gamma(N_{j,1} + \alpha_1)}{\Gamma(N_{j,0} + N_{j,1} + \alpha_0 + \alpha_1)},$$

where d denotes the rules in the list and their order, $d = (L, \{a_j, \theta_j\}_{j=0}^{m})$. Intuitively, one can see that having more of one class and less of the other class will make the likelihood larger. To see this, note that if $N_{j,0}$ is large and $N_{j,1}$ is small (or vice versa) the likelihood for rule j is large.

Let us discuss the prior. There are three terms in the prior, one governing the number of rules m in the list, one governing the size c_j of each rule j (the number of conditions in the rule), and one governing the choice of antecedent condition a_j of rule j given its size. Notation $a_{<j}$ includes the antecedents before j in the rule list if there are any, *e.g.* $a_{<4} = \{a_1, a_2, a_3\}$. Also c_j is the cardinality of antecedent a_j, also written $|a_j|$, as the number of conjunctive clauses in rule a_j. E.g, for rule a being 'x_1=green' and 'x_2<50', this has cardinality 2. $c_{<j}$ includes the cardinalities of the antecedents before j in the rule list. Notation $\mathcal{A}$ is the set of pre-mined antecedents. The prior is:

$$\text{prior}(d|\mathcal{A}, \lambda, \eta) = p(d|\mathcal{A}, \lambda, \eta) = p(m|\mathcal{A}, \lambda) \prod_{j=1}^{m} p(c_j|c_{<j}, \mathcal{A}, \eta) p(a_j|a_{<j}, c_j, \mathcal{A}). \tag{1}$$

The first term is the prior for the number of rules in the list. Here, the number of rules m is Poisson, truncated at the total number of pre-selected antecedents:

$$p(m|\mathcal{A}, \lambda) = \frac{(\lambda^m/m!)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}, \quad m = 0, \ldots, |\mathcal{A}|,$$

where λ is a hyper-parameter. The second term in the prior governs the number of conditions in each rule. The size of rule j is c_j which is Poisson, truncated to remove values for which no rules are available with that cardinality:

$$p(c_j|c_{<j}, \mathcal{A}, \eta) = \frac{(\eta^{c_j}/c_j!)}{\sum_{k \in R_{j-1}(c_{<j}, \mathcal{A})}(\eta^k/k!)}, \quad c_j \in R_{j-1}(c_{<j}, \mathcal{A}),$$

where R_{j-1} is the set of cardinalities available after removing the first $j-1$ rules, and η is a hyperparameter. The third term in the prior governs the choice of antecedent, given that we have determined its size through the second term. We simply have a_j selected from a uniform distribution over antecedents in $\mathcal{A}$ of size c_j, excluding those in $a_{<j}$.

$$p(a_j|a_{<j}, c_j, \mathcal{A}) \propto 1, \quad a_j \in Q_{c_j} = \{a \in \mathcal{A} \setminus \{a_1, a_2, ..., a_{j-1}\} : |a| = c_j\}. \tag{2}$$

As usual, the posterior is the likelihood times the prior.

$$p(d|\mathbf{x}, \mathbf{y}, \mathcal{A}, \alpha, \lambda, \eta) \propto p(\mathbf{y}|\mathbf{x}, d, \alpha)p(d|\mathcal{A}, \lambda, \eta).$$

This is the full model, and the posterior $p(d|\mathbf{x}, \mathbf{y}, \mathcal{A}, \alpha, \lambda, \eta)$ is what we aim to optimize in order to obtain the best rule lists. The hyperparameter λ is chosen by the user to be the desired size of the rule list, and η is chosen as the desired number of terms in each rule. The parameters α_0 and α_1 are usually chosen as 1 in order not to favor one class label over another.

3. Markov Chain Monte Carlo

Given the prior parameters λ, which governs the length of the list, η, which governs the desired number of conditions in the list, and α, which provides a preference over labels (usually we set all the α's to 1), along with the set of pre-mined rules $\mathcal{A}$, the algorithm must select which rules from $\mathcal{A}$ to use, along with their order. We start with a simple update scheme to iteratively maximize the posterior. This could be used for both inference (MCMC) and simulated annealing. We use only MCMC iterates in our experiments, but choose the MAP solution among the iterates. To define a rule list, the algorithm chooses a subset of antecedents from $\mathcal{A}$, along with a permutation of antecedents.

Let us define the neighborhood of a rule list, which is all rule lists that are edit distance 1 away from the initial list. The neighborhood of a rule list can be constructed by removing one rule, adding one rule from $\mathcal{A}$ into the list, or swapping a rule in the list with one that is in $\mathcal{A}$. At each time t, we choose a neighboring rule list at random from the neighborhood by adding, removing, or moving a rule somewhere else within the list. The proposal probabilities for a list d^* from the current list d^t are: $1/[(|d^t|)(|d^t|-1)]$ if the proposal is to move a rule, $1/[(|\mathcal{A}|-|d^t|)(|d^t|+1)]$ for a proposal to add a rule, and $1/|d^t|$ for a proposal to remove a rule.

At each step, we need to evaluate the posterior function on each new rule list. Since this process is repeated many times during the algorithm, speeding up this particular subroutine can have a tremendous increase in computational speed. We improve the speed in three ways: we use high performance language libraries, computational reuse, and theoretical bounds.

4. Implementation Techniques

4.1 Expressing computation as bit vectors

The vast majority of the computational time spent constructing rule sets lies in determining which rules *capture* which observations in a particular rule ordering. As a reminder, for a given ordering of rules in a set, we say that the first rule for which an observation evaluates true captures that observation. The naive implementation of these operations calls for various set operations – checking whether a set contains an element, adding an element to a set, and removing an element from a set. However, set operations are typically slow, and hardware does little to help with efficiency.

We convert all set operations to logical operations on bit vectors, for which hardware support is readily available. The bit vector representation is both memory- and computationally- efficient. The vectors have length equal to the total number of data samples. Before beginning the algorithm, for each rule, we compute the bit vector representing the samples for which the rule generates a true value. For a one million sample data set (or more precisely up to 1,048,576 observations) each rule carries with it 128 KB vector (since a byte consists of 8 bits), which fits comfortably in most L2 caches.

4.2 Representing Intermediate State as Bit Vectors

For each rule list we consider, we maintain similarly sized vectors for each rule in the set indicating which rule in the set captures which observation. Within a rule list, each observation is captured by one and only one rule – the first rule for which the condition evaluates true. Representing the rules and rule lists this way allows us to explore the rule list state space, reusing significant computation. For example, consider a rule list containing m rules. Imagine that we wish to delete rule k from the set. The naive implementation recomputes the "captures" vector for every rule in the set. Our implementation updates only rules $j > k$, using logical operators acting upon the rule list "captures" vector for k, and the rule's "captures" vector for each rule $j > k$. This shortens the run time of the algorithm in practice by approximately 50%.

4.3 An Algebra for Computation Reuse

Our use of bit vectors transforms the large number of set operations performed in a traditional implementation into a set of boolean operations on bit vectors. These are summarized below. In our notation, the rule list contains n rules; k and j are used to represent particular rules in the rule list. Let *k.captures* refer to the captures vector for rule k and *k.init* refer to the original vector associated with each rule, indicating all observations for which the rule evaluates true. Note that *k.captures* $\subset$ *k.init*. Below we show these bit vector operations for the possible MCMC steps.

1. Remove rule k

```
for $j = k + 1$ to $m$ do
    $tmp \leftarrow j.init \vee k.captures$ {Everything $k$ used to capture that could be captured by rule $j$}
    $j.captures \leftarrow j.captures \wedge tmp$ {Add things $k$ use to capture that $j$ can now capture}
    $k.captures \leftarrow k.captures \vee \neg tmp$ {Remove the newly captured items for $j$ from $k$}
end for
```

2. Insert rule into the ruleset at position k

```
$m \leftarrow m + 1$
shift rules $k + 1$ to $m$ up one slot
$captured \leftarrow \{\vec{0_m}\}$
for $j = 1$ to $k - 1$ do
    $captured \leftarrow captured \wedge j.captures$
end for
for $j = k$ to $m$ do
    $j.captures \leftarrow j.init \vee \neg captured$
    $captured \leftarrow j.captures \wedge captured$
end for
```

3. Swap consecutive rules k and j, where $j = k + 1$

```
$captured \leftarrow \{\vec{0_n}\}$
for $k = 1$ to $k - 1$ do
    $captured \leftarrow captured \wedge k.captures$
end for
$j.captures \leftarrow j.init \vee \neg captured$
$k.captures \leftarrow i.captures \vee \neg j.init$
Swap rules $k$ and $j$
```

4. Generalized swap k and j

```
$captured \leftarrow \{\vec{0_n}\}$
for $t = k$ to $j$ do
    $captured \leftarrow captured \wedge k.captures$
end for
swap rules $k$ and $j$
for $t = k$ to $j - 1$ do
    $t.captures \leftarrow captured \vee t.init$
    $captured \leftarrow captured \vee \neg t.captures$
end for
```

4.4 High Performance Bit Manipulation

Having transformed expensive set operations into bit vector operations, we can now leverage both hardware vector instructions and optimized software libraries. We investigated three alternative implementations, each improving computational efficiency from the previous one.

- First, we retained our python implementation with added bit operations.

- Next, we used the python gmpy library to do the population count.

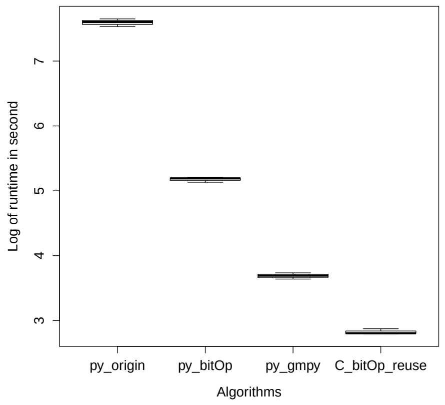

Figure 3: Boxplots of runtime comparison among different implementations. From the original python code, the final code is over two orders of magnitude faster.

- Then, we moved the implementation from Python to C, representing the bit vectors as packed arrays of longs and reusing the information from previous evaluation of the posteriors.

- Finally, we employed GMP library which in practice is slow on small data sets, but faster on big data sets.

To evaluate how each of these steps improved the computation time of the algorithm, we conducted a controlled experiment where each version of the algorithm (corresponding to the three steps above) was given the same data (the UCI adult dataset, divided into three folds), same set of rules, and same number of MCMC iterations (20,000) to run. We created boxplots for the log of the run time over the different folds, which is shown in Figure 3. The code is over two orders of magnitude faster than the original optimized python code.

5. Theoretical Bounds with Practical Implications

We prove two bounds. First we provide an upper bound on the number of rules in a maximum a posteriori rule list. This allows us to narrow our search space to rule lists below a certain size if desired.

Second we provide a branch and bound constraint that eliminates certain prefixes of rule lists if desired. This prevents our algorithm from searching in regions of the space that provably do not contain the maximum a posteriori rule list.

5.1 Upper bound on the number of rules in the list

Given the number of features, the parameter λ for the size of the list, and parameters α_0 and α_1, we can derive an upper bound for the size of a maximum a posteriori rule list. This formalizes how the prior on the number of rules is strong enough to overwhelm the likelihood.

We are considering binary rules and binary features, so the total number of possible rules of each size can be calculated directly. When creating the upper bound, within the proof, we hypothetically exhaust rules from each size category in turn, starting with the smallest sizes. We discuss this further below.

Let $|Q_c|$ be the number of rules that remain in the pile that have c logical conditions. The sequence of b's that we define next is a lower bound for the possible sequence of $|Q_c|$'s. In particular, b represents the sequence of sizes of rules that would provide the smallest possible $|Q_c|$. Intuitively, the sequence of b's arises when we deplete the rules of size 1, then deplete all of the rules of size 2, etc. The number of ways to do this is given exactly by the b values, computed as follows.

Definition Let P be the number of features, and $\mathbf{b} = \left\{b_0, b_1, b_2, ...b_{2^P-1}\right\}$ be a vector of length P defined as follows:

```
index = 0
$b_0$=1
for $c = 0$ to $\left\lfloor \frac{P}{2} \right\rfloor$ do

  for $j = \binom{P}{c}$ down to 1 (using step size=-1) do
    index = index + 1
    $b_{\text{index}}$ = j
  end for

  if (c+c != $P$) then

    for $j = \binom{P}{P-c}$ down to 1 (using step size = -1) do
      index = index + 1
      $b_{\text{index}}$ = j
    end for
  end if
end for
```

Figure 4 is an illustration for the b_j's when the number of features is $P = 5$.

We will use the b's within the theorem below. In our notation, rule list d is defined by the antecedents and the probabilities on the right side of the rules, $d = (m, \{a_l, \theta_l\}_{l=1}^m)$.

Theorem 1 *The size m^* of any MAP rule list d^* (with parameters λ, η, and $\alpha = (\alpha_0, \alpha_1)$) obeys $m^* \leq m_{max}$, where*

$$m_{\max} = \min\left\{2^P - 1, \max\left\{m' \in \mathbb{Z}_+ : \frac{\lambda^{m'}}{m'!} \geq \frac{\Gamma(N_- + \alpha_0)\Gamma(N_+ + \alpha_1)}{\Gamma(N + \alpha_0 + \alpha_1)} \prod_{j=1}^{m'} b_j\right\}\right\}. \quad (3)$$

In the common parameter choice $\alpha_0 = 1$ and $\alpha_1 = 1$, this reduces to:

$$m_{\max} = \min\left\{2^P - 1, \max\left\{m' \in \mathbb{Z}_+ : \frac{\lambda^{m'}}{m'!} \geq \frac{\Gamma(N_- + 1)\Gamma(N_+ + 1)}{\Gamma(N + 2)} \prod_{j=1}^{m'} b_j\right\}\right\}. \quad (4)$$

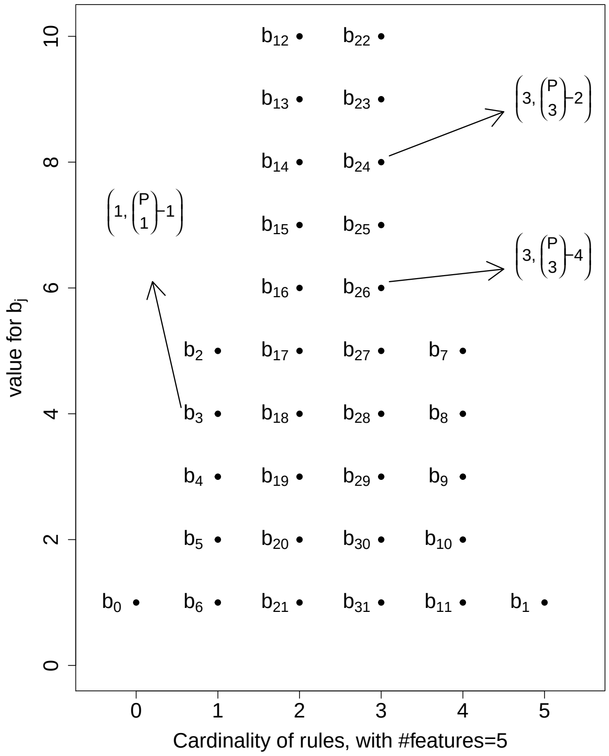

Figure 4: Example of b_j's for a 5-feature dataset.

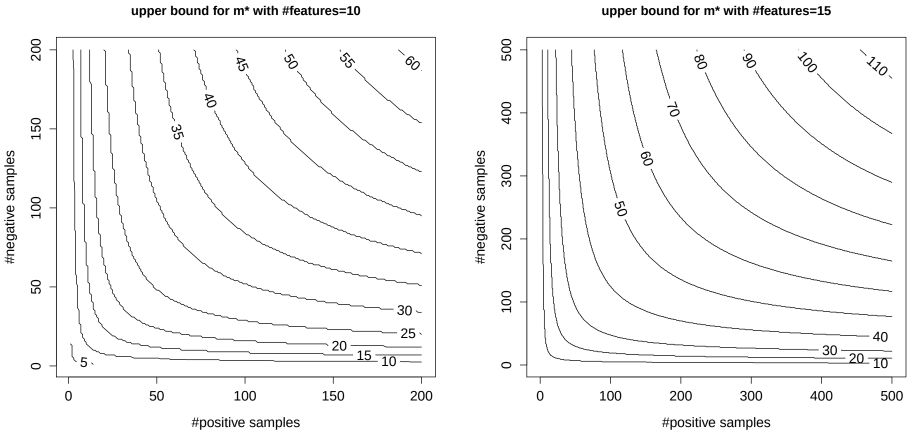

Figure 5: Upper bound from Theorem 1 for the length of the rule list when the number of features is 10 (left figure) and 15 (right figure).

The proof is in the appendix.

Figure 5 illustrates the use of this theorem. In particular, we plotted the upper bound for m^* from the Theorem 1 when the number of features P is 10 in Figure 5 (left) and we plotted the upper bound when the number of features is 15 in Figure 5 (right), with $\lambda = 3$ and $\alpha_0 = \alpha_1 = 1$. For instance, when there are 10 features (left plot) and approximately 100 positive and 100 negative observations, there will be at most about 36 rules.

5.2 Prefix Bound

We next provide a bound that eliminates certain regions of the rule space from consideration. Consider a rule list beginning with rules $a_1, .., a_p$. If the best possible rule list starting with $a_1, .., a_p$ cannot beat the posterior of the best rule list we have found so far, then we know any rule list starting with $a_1, .., a_p$ is suboptimal. In that case, we should stop exploring rule lists starting with $a_1, .., a_p$. This is a type of branch and bound strategy, in that we have now eliminated (bounded) the entire set of lists starting with $a_1, .., a_p$. We formalize this intuition below.

Denote the rule list at iteration t by $d^t = (a_1^t, a_2^t, ..., a_{m_t}^t, a_0)$. The current best posterior probability has value v_t^*, that is

$$v_t^* = \max_{t' \leq t} \text{Posterior}(d^{t'}, \{(x_i, y_i)\}_{i=1}^n).$$

Let the current rule list be $d = (a_1, a_2, ...a_m, a_0)$. Let d_p denote a prefix of length p of the rule list d, i.e., $d_p = (a_1, a_2, ...a_p)$, where $a_1, a_2, ..., a_p$ is the same as the first p rules in d. Figure 6 illustrates this notation. We want to determine whether a rule list starting with d_p could be better than the best we have seen so far. Define $\Upsilon(d_p, \{(x_i, y_i)\}_{i=1}^n)$ as follows:

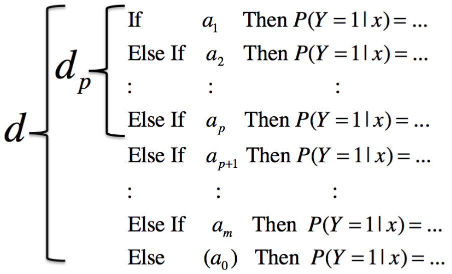
d
d_p
If a_1 Then $P(Y=1|x)=...$
Else If a_2 Then $P(Y=1|x)=...$
: : :
Else If a_p Then $P(Y=1|x)=...$
Else If a_{p+1} Then $P(Y=1|x)=...$
: : :
Else If a_m Then $P(Y=1|x)=...$
Else (a_0) Then $P(Y=1|x)=...$

Figure 6: Notation for a rule list d and its prefix d_p.

$$
\begin{aligned}
&\Upsilon(d_p, \{(x_i, y_i)\}_{i=1}^n) \\
&:= \frac{\lambda^{\max(p,\lambda)}/(\max(p,\lambda))!}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)} \left(\prod_{j=1}^{p} p(c_j|c_{<j}, \mathcal{A}, \eta)\frac{1}{|Q_{c_j}|}\right) \times \\
&\quad \left(\prod_{j=0}^{m} \frac{\Gamma(N_{j,0}+1)\Gamma(N_{j,1}+1)}{\Gamma(N_{j,0}+N_{j,1}+2)}\right) \frac{\Gamma(1+N_0-\sum_{j=1}^{p} N_{j,0})}{\Gamma(2+N_0-\sum_{j=1}^{p} N_{j,0})} \frac{\Gamma(1+N_1-\sum_{j=1}^{p} N_{j,1})}{\Gamma(2+N_1-\sum_{j=1}^{p} N_{j,1})}.
\end{aligned}
$$

Here, $N_{j,0}$ is the number of points captured by rule j with label 0, and $N_{j,1}$ is the number of points captured by rule j with label 1,

$$N_{j,0} = |\{i : \text{Captr}(i) = j \text{ and } y_i = 0\}|, \quad N_{j,1} = |\{i : \text{Captr}(i) = j \text{ and } y_i = 1\}|.$$

The result states that for a rule list with prefix d_p, if the upper bound on the posterior, $\Upsilon(d_p)$, is not as high as the posterior of the best rule list we have seen so far, then d_p is a bad prefix, which cannot lead to a MAP solution. It tells us we no longer need to consider rule lists starting with d_p.

Theorem 2 *For rule list $d = \{d_p, a_{p+1}, ..., a_m, a_0\}$, if*

$$\Upsilon(d_p, \{(x_i, y_i)\}_{i=1}^n) < v_t^*,$$

then for $\alpha_0 = 1$ and $\alpha_1 = 1$, we have

$$d \notin \text{argmax}_{d'}\text{Posterior}(d', \{(x_i, y_i)\}_{i=1}^n). \tag{5}$$

Theorem 2 is implemented in our code in the following way: for each random restart, the initial rule in the list is checked against the bound of Theorem 2. If the condition $\Upsilon(d_1) < v_t^*$ holds, we throw out this initial rule and choose a new one, because that rule provably cannot be the first rule in an optimal rule list. Theorem 2 provides a substantial computational speedup in finding high quality or optimal solutions. In some cases, it provides a full order of magnitude speedup. Because it has been so useful in practice, we provide illustrative examples.

5.3 Demonstrations of Theorem 2

Demonstration 1: We use the Tic Tac Toe dataset from the UCI repository (see Bache & Lichman, 2013). Each observation is a tic tac toe board after the game has ended. If the X player wins, the label of the observation is 1, otherwise it is 0. Let us consider a rule list starting with the following two rules:

If

	o	

then ...

else if

		o

then ...

else if ...

The first rule says that the board contains an "O" in the bottom middle spot, and the rule says nothing about other spots. Intuitively this is a particularly bad rule, since it captures a lot of possible tic tac toe boards, and on its own, cannot distinguish between winning and losing boards for the "X" player. Similarly, the second rule also does not discriminate well. Thus, we expect any rule list starting with these two rules to perform poorly. We can show this using the theorem. On one of three folds of the data, this rule list has a log posterior that is upper bounded at -272.51. From an earlier run of the algorithm, we know there is a rule list with a posterior of -105.012. (That rule list is provided in Table 2 and contains exactly one rule for each way the "X" player could have three X's in a row on the board.) Since the upper bound on the posterior for this rule list (-272.51) is less than -105.012, there does not exist an optimal rule list starting with these two rules.

Demonstration 2: In contrast with Demonstration 1, a rule list starting as follows cannot be excluded.

If

o	o	o

then ...

else if

o		
o		
o		

then ...

else if ...

These first two rules says that the "O" player has three O's in a row, which means the "X" player could not have won. This prefix has a log posterior that is upper bounded at -35.90, which is higher than than -105.012. Thus we cannot exclude this prefix as being part of an optimal solution. As it turns out, there are high posterior solutions starting with this prefix. One such solution is shown in Table 3 below.

6. Experiments

We provide a comparison of algorithms along three dimensions: solution quality (AUC - area under the ROC curve), sparsity, and scalability. Sparsity will be measured as the number of leaves in a decision tree or as the number of rules in a rule list. Scalability will be measured in computation time. SBRL tends to achieve a useful balance between these three quantities.

Let us describe the experimental setup. As baselines, we chose popular classification algorithms to represent the sets of uninterpretable methods and the set of "interpretable" methods. To represent the class of uninterpretable methods, we chose logistic regression, SVM RBF, random forests (RF), and boosted decision trees (ADA). None of these methods are designed to yield sparse classifiers. They are designed to yield scalable and accurate classifiers. To represent the class of "interpretable" greedy splitting algorithms, we chose CART and C4.5. CART tends to yield sparse classifiers, whereas C4.5 tends to be much less interpretable. Other experiments (see Letham et al., 2015; Wang & Rudin, 2015b) have accuracy/interpretability comparisons to Bayesian Rule Lists and Falling Rule Lists, so our main effort here will be to add the scalability component. We benchmark using publicly available datasets after some data pre-processing(for example, using quantiles instead of real-valued variables and merging some discrete levels together to avoid too many levels in one column):

- the Tic Tac Toe dataset (see Bache & Lichman, 2013), where the goal is to determine whether the "X" player wins (this is easy for a human who would check for three X's in a row),
- the Adult dataset (see Bache & Lichman, 2013), where we aim to predict whether an individual makes over $50K in a year,
- the mushroom dataset (see Bache & Lichman, 2013), where the goal is to predict whether a mushroom is poisonous,
- the nursery dataset (see Bache & Lichman, 2013), where the goal is to predict whether a child's application to nursey school will be in either the "very recommended" or "special priority" categories,
- the Telco customer churn dataset (see WatsonAnalytics), where the goal is to predict whether a customer will leave the service provider,
- the Titanic dataset (see Bache & Lichman, 2013), where the goal is to predict who survived the sinking of the Titanic.

Evaluations of prediction quality, sparsity, and timing were done using 3-fold cross validation.

For creating the random starting rule list, the initial rule length was set to 1 rule (not including the default rule). The minimum and maximum rule size for the rule-mining algorithm were set at 1 and 2, respectively, except for the Tic Tac Toe dataset, for which the maximum was set to 3. Because of the nature of the Tic Tac Toe dataset, it is more interpretable to include rules of size 3 than to exclude them. For rule mining, we chose the

Run Time	LR	SVM	CART	C4.5	RF	ADA	SBRL
CV1	0.040	0.156	0.023	0.141	0.290	1.330	0.579
CV2	0.043	0.170	0.024	0.185	0.380	1.346	0.596
CV3	0.045	0.284	0.052	0.132	0.481	1.329	0.538

Table 1: Run time on Tic Tac Toe dataset.

minimum support of rules from (5%, 10%, 15%, etc.) so that the total number of rules was approximately 300.

The prior parameters were fixed at $\eta = 1$, and $\alpha = (1, 1)$. For the λ for each dataset, we first let λ be 5, and ran SBRL once with the above parameters. Then we fixed λ at the length of the returned rule list for that dataset. It is possible that the solution quality would increase if SBRL was run for a larger number of iterations. For the purpose of providing a controlled experiment, the number of iterations was fixed at 5,000 for each chain of the 11 chains of SBRL, which we ran in series on a laptop. For some of the datasets, we ran SBRL for 50,000 iterations (rather than 5,000) to illustrate additional solutions. Every time SBRL started a new rule list, we checked the initial rule in the list to see whether the upper-bound on its posterior (by Theorem 2) was greater than the best rule list we have found so far. If not, the rule was replaced until the condition was satisfied.

Results for the Tic Tac Toe dataset are shown in Figure 7, Figure 8 and Table 1. Each observation in this dataset is a tic tac toe board after the game has finished. If there are 3 X's in a row, the label of the board is 1, otherwise 0. This should not be a difficult learning problem since there are solutions with perfect accuracy on the training set that generalize to the test set. However, none of the other machine learning methods can find one of these perfect solutions. All of the other methods make heavy approximations, whether it is linear approximations (logistic regression, boosting), or greedy splitting (random forests, CART, C4.5); experiments on this dataset show that sometimes there are severe sacrifice made for those approximations, whether it is in terms of accuracy or in terms of sparsity. Figure 7 shows the sacrifice in AUC made by CART and C4.5. The other methods (RF, SVM, logistic regression, ADA) do not give sparse solutions. In this figure, the algorithms were all used in their default modes, using their own internal cross-validation routines.

Figure 8 delves further on the decision tree and SBRL models to illustrate the AUC/sparsity tradeoff. It shows a scatter plot of AUC vs. number of leaves, where each point represents an evaluation of one algorithm, on one fold, with one parameter setting. For SBRL, there was no parameter tuning, so there are are three points, one for each of the three folds. We tried many different parameter settings for CART (in blue), and many different parameter settings for C4.5 (in gray), none of which were able to achieve points on the efficient frontier defined by the SBRL method.

This is an example where the SBRL model far surpasses its competitors. Tables 2, 3, and 4 show the models from the 3 BRL folds. SBRL's run time was on average around 0.6 seconds.

For the Adult dataset, results are in Figure 9, Figure 10 and Table 5. Adult contains 45,121 observations and 12 features, where each observation is an individual, and the features are census data, including demographics, income levels, and other financial infor-

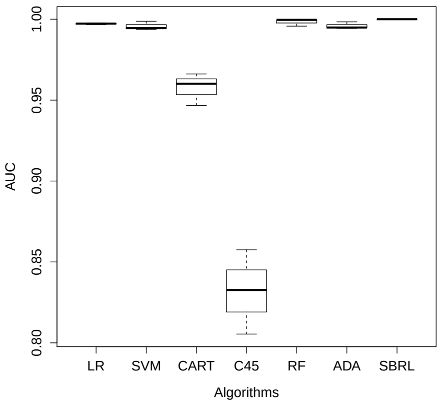

Figure 7: Comparison of AUC of ROC among different methods on Tic Tac Toe dataset.

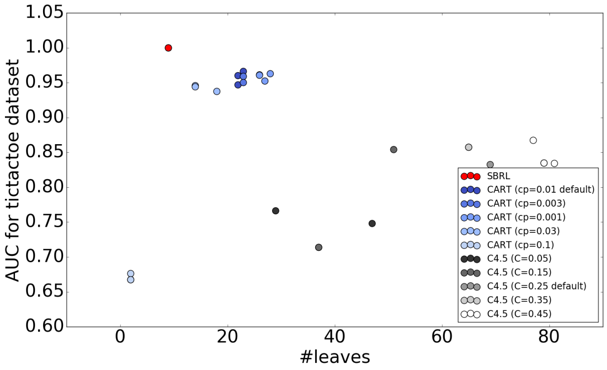

Figure 8: Scatter plot of AUC against the number of leaves (sparsity) for Tic Tac Toe dataset. All folds are included, along with results from several different settings of CART and C4.5's parameters.

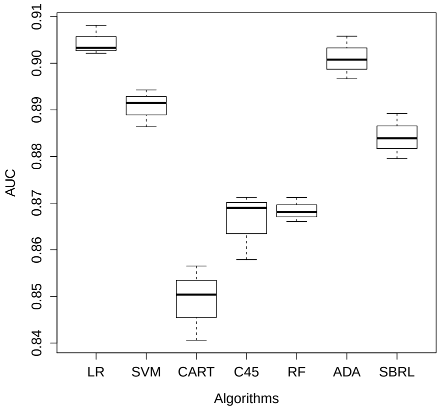

Figure 9: Comparison of AUC among different methods on the Adult dataset.

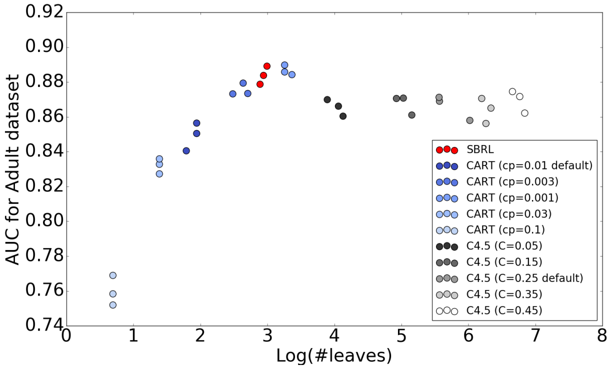

Figure 10: Scatter plot of AUC against the number of leaves (sparsity) for the Adult dataset. All folds are included, along with results from several different settings of CART and C4.5's parameters.

Rule-list	Antecedent risk
if (x3&x7&x5),	0.98
else if (x1&x9&x5),	0.98
else if (x8&x2&x5),	0.98
else if (x6&x3&x9),	0.98
else if (x4&x6&x5),	0.98
else if (x2&x1&x3),	0.98
else if (x8&x7&x9),	0.98
else if (x4&x1&x7),	0.98
else (default),	0.0044

Table 2: Example of rule list for Tic Tac Toe dataset, fold 1 (CV1).

Rule-list	Test Accuracy	Antecedent risk
if (o9&o1&o5),	1.00	0.03
else if (o7&o3&o5),	1.00	0.026
else if (o6&o9&o3),	1.00	0.037
else if (o8&o2&o5),	1.00	0.036
else if (o4&o6&o5),	1.00	0.04
else if (o8&o9&o7),	1.00	0.04
else if (o2&o1&o3),	1.00	0.03
else if (o4&o7&o1),	1.00	0.04
else (default),	1.00	0.97

Table 3: Example of rule list for Tic Tac Toe dataset, fold 2 (CV2).

Rule-list	Antecedent risk	Test Accuracy
if (x8&x9&x7),	0.98	1.00
else if (x6&x9&x3),	0.98	1.00
else if (o4&o1&o7),	0.042	1.00
else if (o6&o3&o9),	0.043	1.00
else if (x4&x7&x1),	0.98	1.00
else if (x2&x1&x3),	0.98	1.00
else if (o2&o3&o1),	0.050	1.00
else if (o5),	0.0079	1.00
else if (o3&x7),	0.71	1.00
else if (o8&o7&o9),	0.040	1.00
else (default),	0.99	0.96

Table 4: Example of rule list for Tic Tac Toe dataset, fold 3 (CV3).

mation. Here, SBRL, which was untuned and forced to be sparse, performed only slightly worse than several of the uninterpretable methods. Its AUC performance dominated those of the CART and C4.5 algorithms. As the scatter plot shows, even if CART were tuned on the test set, it would have performed at around the same level, perhaps slightly worse than SBRL. The timing for SBRL was competitive, at 41 to 44 seconds, where 35 seconds were

Run Time	LR	SVM	CART	C4.5	RF	ADA	SBRL
CV1	1.582	141.894	0.653	0.459	19.370	33.359	11.475
CV2	1.115	139.857	0.692	0.488	19.981	33.428	11.042
CV3	1.112	138.956	0.633	0.467	18.866	32.986	10.948

Table 5: Run Time on Adult dataset

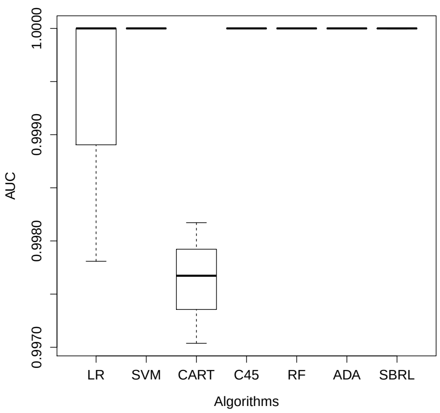

Figure 11: Comparison of AUC of ROC among different methods on mushroom dataset.

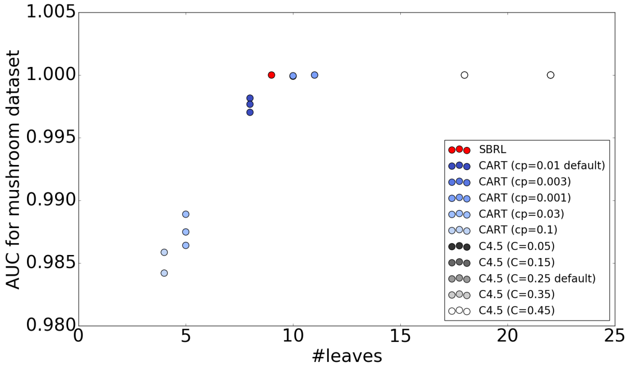

Figure 12: Scatter plot of AUC against the number of leaves (sparsity) for mushroom dataset. All folds are included, along with results from several different settings of CART and C4.5's parameters.

Run Time	LR	SVM	CART	C4.5	RF	ADA	SBRL
CV1	1.488	2.640	0.098	0.320	1.558	7.073	6.953
CV2	1.508	2.934	0.124	0.328	1.593	7.029	6.829
CV3	1.421	2.891	0.106	0.334	1.632	6.885	7.056

Table 6: Run time on mushroom dataset

MCMC iterations. If the chains were computed in parallel rather than in series, it would speed up computation further. Figure 2 contains one of the rule lists we produced.

For the mushroom dataset, results are shown in Figure 11, Figure 12 and Table 6. Perfect AUC scores were obtained using all the methods we tried, with the exception of untuned CART. On the scatterplot within Figure 12, there are several solutions with perfect accuracy found by SBRL, tuned CART and and C4.5, of sizes between 9 and 22 rules. SBRL found a solution of size 9 after 5,000 iterations. CART also found a solution of size 10 after tuning. One thing worth mentioning is that using the original dataset without any data preprocessing, SBRL found a solution of size 7 after 50000 iterations and CART found a solution of size 6. The difference in posterior values between the perfect solutions of similar size was extremely small because of our choice of (untuned) λ. (Tuning and smaller choices for λ would improve computation.) Figure 1 contains one of the rule lists we produced. The CART tree and rule lists produced for this dataset look entirely different. This is because SBRL views each categorical feature as a separate binary variable, whereas CART does not; it can split arbitrarily on categorical variables without penalty. If we had done additional preprocessing on the features to create more splits, we could potentially get rule lists that look like CART's tree. What we got were totally different, yet almost equally perfect, solutions.

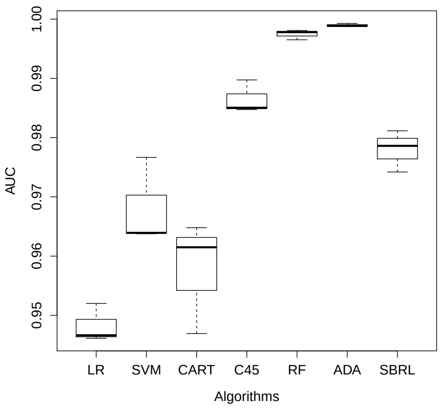

Figure 13: Comparison of AUC of ROC among different methods on nursery dataset.

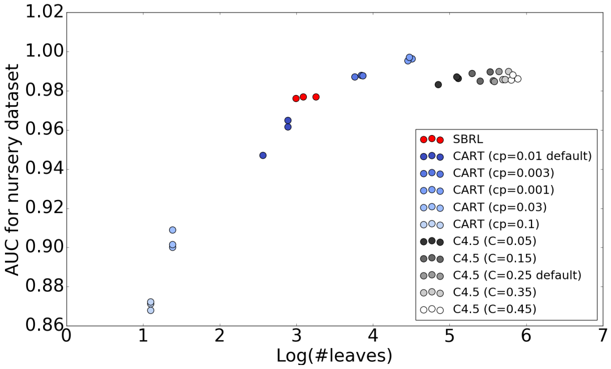

Figure 14: Scatter plot of AUC against the number of leaves (sparsity) for nursery dataset. All folds are included, along with results from several different settings of CART and C4.5's parameters.

Run Time	LR	SVM	CART	C4.5	RF	ADA	SBRL
CV1	0.315	11.406	0.396	0.174	3.873	6.939	3.618
CV2	0.349	7.962	0.114	0.188	3.052	8.185	3.558
CV3	0.960	10.976	0.109	0.203	2.949	6.912	3.697

Table 7: Run Time of nursery dataset

The results from the nursery dataset are shown in Figure 13, Figure 14 and Table 7. A similar story holds as for the previous datasets: SBRL is on the optimal frontier of accuracy/sparsity without tuning and with reasonable run time.

Figure 15, Figure 16 and Table 8 show the results for the Telco dataset, which contains 7043 observations and 18 features. Similar observations hold for this dataset. The models from the three folds are provided in Tables 9, 10 and 11. These models illustrate that generally, rule lists are not the same between folds, but often tend to use similar rules.

The Titanic dataset evaluation results are in Figures 17, 18 and Table 12, which contains data about 2201 passengers and crew aboard the Titanic.

The results on all of these datasets are consistent. On each dataset, SBRL produces results that are reliable (unlike CART) and sparse (unlike C4.5). SBRL was used untuned

Run Time	LR	SVM	CART	C4.5	RF	ADA	SBRL
CV1	0.240	4.776	0.132	0.256	2.937	6.610	2.325
CV2	0.343	4.771	0.125	0.249	2.955	6.598	2.425
CV3	0.254	4.871	0.135	0.224	3.559	6.738	2.378

Table 8: Run Time of Telco-Customer-Churn dataset

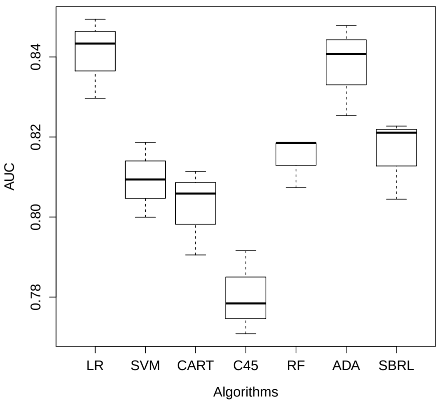

Figure 15: Comparison of AUC of ROC among different methods on Telco dataset.

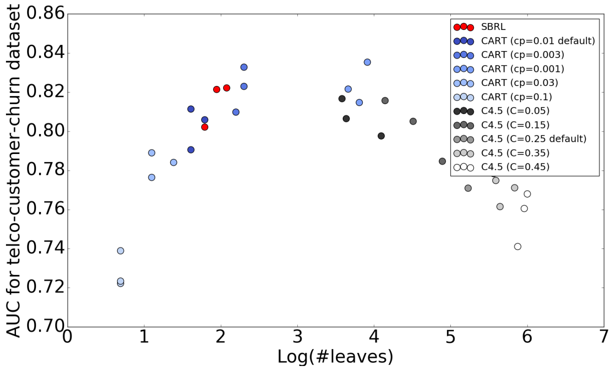

Figure 16: Scatter plot of AUC against the number of leaves (sparsity) for Telco dataset. All folds are included, along with results from several different settings of CART and C4.5's parameters.

Rule-list	Risk	Test Accuracy
if (Contract=One_year &StreamingMovies=Yes),	0.20	0.82
else if (Contract=One_year),	0.050	0.96
else if (tenure<1year &InternetService=Fiber_optic),	0.70	0.71
else if (Contract=Two_year),	0.029	0.97
else if (InternetService=Fiber_optic &OnlineSecurity=No),	0.48	0.58
else if (OnlineBackup=No &TechSupport=No),	0.41	0.61
else (default),	0.22	0.78

Table 9: example of rule list for Telco-Customer-Churn dataset fold 1 (CV1).

Rule-list	Risk	Test Accuracy
if (Contract=One_year &StreamingMovies=Yes),	0.20	0.81
else if (tenure<1year &InternetService=Fiber_optic),	0.70	0.70
else if (tenure<1year &OnlineBackup=No),	0.44	0.57
else if (InternetService=Fiber_optic &Contract=Month-to-month),	0.43	0.57
else if (Contract=Month-to-month),	0.22	0.82
else (default),	0.034	0.97

Table 10: Example of rule list for Telco-Customer-Churn dataset fold 2 (CV2).

Rule-list	Risk	Test Accuracy
if (Contract=One_year&StreamingMovies=Yes),	0.20	0.81
else if (Contract=Two_year),	0.032	0.98
else if (Contract=One_year),	0.054	0.97
else if (tenure<1year&InternetService=Fiber_optic),	0.70	0.72
else if (PaymentMethod=Electronic_check &InternetService=Fiber_optic),	0.48	0.45
else (TechSupport=No&OnlineSecurity=No),	0.42	0.64
else (default),	0.22	0.78

Table 11: example of rule list for Telco-Customer-Churn dataset CV3.

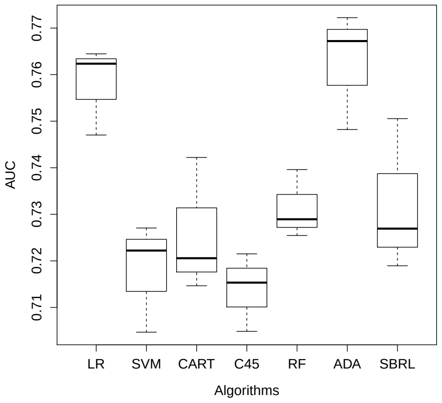

Figure 17: Comparison of AUC of ROC among different methods on titanic dataset.

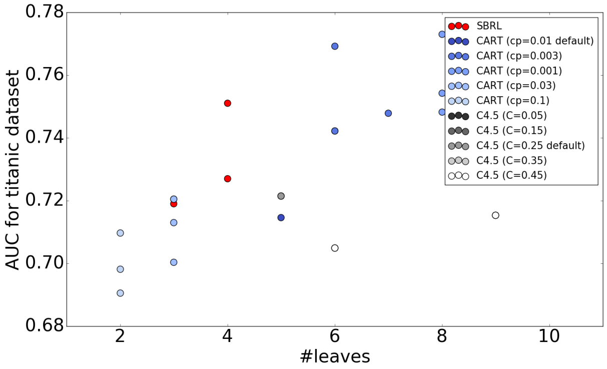

Figure 18: Scatter plot of AUC against the number of leaves (sparsity) for titanic dataset. All folds are included, along with results from several different settings of CART and C4.5's parameters.

Run Time	LR	SVM	CART	C4.5	RF	ADA	SBRL
CV1	0.018	0.273	0.022	0.099	0.279	1.178	0.351
CV2	0.017	0.318	0.016	0.086	0.367	1.175	0.366
CV3	0.019	0.324	0.016	0.085	0.297	1.166	0.373

Table 12: Run Time of titanic dataset

throughout these experiments. The quality of predictions was between the interpretable methods the best-in-hindsight black-box methods, almost always outperforming the interpretable methods, and often outperforming some of the uninterpretable methods. The run times are longer but still reasonable, and also adjustable since the user can pre-determine exactly how long to run the method.

7. Related Works and Discussion

Rule lists are not very different from decision trees in capacity; any decision tree can be made into a decision list simply by placing a rule in the list to represent each leaf of the decision tree. Rule lists are automatically a type of decision tree. Thus this method is really a direct competitor for CART.

The algorithm proposed here strikes a balance between accuracy, scalability, and interpretability. Interpretability has long since been a fundamental topic in artificial intelligence (see Rüping, 2006; Bratko, 1997; Dawes, 1979; Vellido, Martín-Guerrero, & Lisboa, 2012; Giraud-Carrier, 1998; Holte, 1993; Shmueli, 2010; Huysmans, Dejaeger, Mues, Vanthienen, & Baesens, 2011; Freitas, 2014). Because the rule lists created by our method are designed to be interpretable, one would probably not want to boost them, or combine them in other ways to form more complicated models. This contrasts with, for instance, Friedman and Popescu (2008), who linearly combine pre-mined rules.

This work enables us to globally control decision trees in a sense, which could lead to more interesting styles of trees, and different forms of interpretability. For example, one cannot easily construct a Falling Rule List with a greedy splitting method, but can construct one with a global optimization approach. A Falling Rule List Wang and Rudin (2015b) is a decision list where the probabilities of success decrease as we descend along the list. This means we can target the highest probability subgroup by checking only a few conditions. A Causal Falling Rule List (CFRL) Wang and Rudin (2015a) is another such example. These model causal effects (conditional differences) rather than outcomes. The first rule in the list pinpoints the subgroup with the largest treatment effect. It is possible that many other exotic types of constrained models could be constructed in a computationally efficient way using the ideas in this paper. One could go beyond logical models and consider also mixed logical/linear models (see Wang, Fujimaki, & Motohashi, 2015a).

Rule lists and their variants are currently being used for text processing (King, Lam, & Roberts, 2014), discovering treatment regimes (Zhang, Laber, Tsiatis, & Davidian, 2015), and creating medical risk assessments (Letham et al., 2015; Souillard-Mandar, Davis, Rudin, Au, Libon, Swenson, Price, Lamar, & Penney, 2015), among other applications.

There are other subfields where one would pre-mine rules and use them in a classifier. Inductive logic programming (Muggleton & De Raedt, 1994), greedy top-down decision list algorithms (Rivest, 1987; Sokolova, Marchand, Japkowicz, & Shawe-Taylor, 2003; Anthony, 2005; Marchand & Sokolova, 2005; Rudin, Letham, & Madigan, 2013; Goessling & Kang, 2015), associative classification (Vanhoof & Depaire, 2010; Liu, Hsu, & Ma, 1998; Li, Han, & Pei, 2001; Yin & Han, 2003) and its Bayesian counterparts (McCormick, Rudin, & Madigan, 2012) all fall into this category. None of the methods in these fields follow the same general procedure as we do, where rules are fully optimized into an optimal tree using low-level computations, and where rules are eliminated based on theoretical motivation, as we have in Sections 5.

Teleo-reactive programs (Nilsson, 1994) use a decision list structure and could benefit from learning this structure from data.

There are a series of works from the mid-1990's on finding optimal decision trees using dynamic programming and search techniques (e.g., Bennett & Blue, 1996; Auer, Holte, & Maass, 1995; Dobkin, Fulton, Gunopulos, Kasif, & Salzberg, 1996), mainly working with only fixed depth trees. None of these works use the systems level techniques we use to speed up computation. Farhangfar, Greiner, and Zinkevich (2008) use a screening step that reduces the number of features, using the Naïve Bayes assumption that the features are independent, given the class, and then uses dynamic programming to construct an optimal fixed-depth tree. One particularly interesting work following this literature is that of Nijssen and Fromont (2010), which allows for pre-mined rules to form trees, but in a different way than our method or associative classifiers. Nijssen and Fromont (2010) has the user pre-mine all possible *leaves*, enumerating all conditions leading to that leaf. (By contrast, in our work and in associative classification, we mine only small conjunctions, and their ordered combination creates leaves.) Nijssen and Fromont (2010) warn about issues related to running out of memory. As a possible extension, the work proposed here could be modified to handle regularized empirical risk minimization, in particular it could use the objective of Rudin and Ertekin (2015), which is a balance between accuracy and sparsity of rule lists. It could also be modified to handle disjunctive normal form classifiers, for which there are now Bayesian models analogous to the ones studied in this work (Wang, Rudin, Doshi, Liu, Klampfl, & MacNeille, 2015b). Bayesian tree models may also be able to be constructed using our setup, where one would mine rules and create a globally optimal tree (Dension, Mallick, & Smith, 1998; Chipman, George, & McCulloch, 2002, 2010). It may be logistically more difficult to code trees than lists in order to take advantage of the fast lower level computations, but this is worth further investigation.

A theoretical result of Rudin et al. (2013) states that the VC (Vapnik-Chervonenkis) dimension of the set of rule lists created using pre-mined rules is exactly the size of the set of pre-mined rules. This provides a connection to linear models, whose complexity is the number of features plus 1. That is, the VC dimension of rule lists created from $|\mathcal{A}|$ predefined rules is essentially the same as that of linear models with $|\mathcal{A}|$ features. If some rules are eliminated (for instance based on the theorems in Section 5) then the VC dimension is the size of the set of rules that remain.

Conclusion

We finish by stating why/when one would want to use this particular method. SBRL is not meant as a competitor for black box classifiers like neural networks, support vector machines, gradient boosting or random forests. It is useful when machine learning tools are used as a decision aid to humans, who need to understand the model in order to trust it and make data-driven decisions. SBRL is not a greedy splitting/pruning procedure like decision tree algorithms (CART, C4.5), which means that it more reliably computes high quality solutions, at the possible expense of additional computation time. Many of the decision tree methods do not compute sparse trees, and do not provide interpretable models, as we have seen with C4.5. Our code is a strict improvement over the original Bayesian Rule Lists algorithm if one is looking for a maximum a posteriori solution. It is faster because of careful use of low level computations and theoretical bounds.

Acknowledgments

The authors would like to acknowledge partial funding provided by Philips, Wistron, and Siemens.

Code

Code for SBRL is available at the following link: https://github.com/Hongyuy/sbrlmod

References

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

Auer, P., Holte, R. C., & Maass, W. (1995). Theory and applications of agnostic pac-learning with small decision trees.. pp. 21–29. Morgan Kaufmann.

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

Bennett, K. P., & Blue, J. A. (1996). Optimal decision trees. Tech. rep., R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute.

Bratko, I. (1997). Machine learning: between accuracy and interpretability. In Della Riccia, G., Lenz, H.-J., & Kruse, R. (Eds.), *Learning, Networks and Statistics*, Vol. 382 of *International Centre for Mechanical Sciences*, pp. 163–177. Springer Vienna.

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

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

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

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

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

Farhangfar, A., Greiner, R., & Zinkevich, M. (2008). A fast way to produce optimal fixed-depth decision trees. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2008), Fort Lauderdale, Florida, USA, January 2-4, 2008.*

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

Friedman, J. H., & Popescu, B. E. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, *2*(3), 916–954.

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

Goessling, M., & Kang, S. (2015). Directional decision lists. ArXiv e-prints 1508.07643.

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

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

King, G., Lam, P., & Roberts, M. (2014). Computer-assisted keyword and document set discovery from unstructured text. Tech. rep., Harvard.

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

Li, W., Han, J., & Pei, J. (2001). CMAR: accurate and efficient classification based on multiple class-association rules. In *IEEE International Conference on Data Mining*, pp. 369–376.

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

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

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

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

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

Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

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

Rudin, C., & Ertekin, S. (2015). Learning optimized lists of classification rules. Tech. rep., Massachusetts Institute of Technology.

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

Rüping, S. (2006). *Learning interpretable models*. Ph.D. thesis, Universität Dortmund.

Shmueli, G. (2010). To explain or to predict?. *Statistical Science*, *25*(3), 289–310.

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

Souillard-Mandar, W., Davis, R., Rudin, C., Au, R., Libon, D. J., Swenson, R., Price, C. C., Lamar, M., & Penney, D. L. (2015). Learning classification models of cognitive conditions from subtle behaviors in the digital clock drawing test. *Machine Learning, First Online*, 1–49. Accepted.

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

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

Wang, F., & Rudin, C. (2015a). Causal falling rule lists. Working Paper.

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

Wang, J., Fujimaki, R., & Motohashi, Y. (2015a). Trading interpretability for accuracy: Oblique treed sparse additive models. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1245–1254.

Wang, T., Rudin, C., Doshi, F., Liu, Y., Klampfl, E., & MacNeille, P. (2015b). Bayesian or's of and's for interpretable classification with application to context aware recommender systems. Tech. rep., Massachusetts Institute of Technology.

WatsonAnalytics, I. https://community.watsonanalytics.com/telco-customer-churn/.

Yin, X., & Han, J. (2003). Cpar: classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, ICDM '03, pp. 331–335.

Zhang, Y., Laber, E., Tsiatis, A., & Davidian, M. (2015). Using decision lists to construct interpretable and parsimonious treatment regimes. ArXiv e-prints, 1504.07715.

Appendix: Proof of Theorem 1

To prove this, we will show that any rule list with more than $m_{\max}$ rules has a lower posterior than the trivial empty rule list. This means any rule list with more than $m_{\max}$ terms cannot be a MAP rule list. Denote ϕ as the trivial rule list with only the default rule. By definition of d^* as a MAP rule list, it has a posterior at least as high as ϕ.

$$\text{Posterior}(d^*|\mathcal{A}, X, Y, \alpha, \lambda, \eta) \geq \text{Posterior}(\phi|\mathcal{A}, X, Y, \alpha, \lambda, \eta)$$

$$\frac{(\lambda^{m^*}/m^*!)}{\sum\limits_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)} \prod_{j=0}^{m^*} \left(\frac{(\eta^{c_j}/c_j!)}{\sum\limits_{k \in R_{j-1}(c_{<j}, \mathcal{A})} (\eta^k/k!)} \frac{1}{|Q_{c_j}|} \frac{\Gamma(N_{j,0}+\alpha_0)\Gamma(N_{j,1}+\alpha_1)}{\Gamma(N_{j,0}+N_{j,1}+\alpha_0+\alpha_1)} \right)$$

$$\geq \frac{\lambda^0/0!}{\sum\limits_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)} \frac{\Gamma(N_- + \alpha_0)\Gamma(N_+ + \alpha_1)}{\Gamma(N+\alpha_0+\alpha_1)}$$

$$\frac{\lambda^{m^*}}{m^*!} \geq \frac{\Gamma(N_- + \alpha_0)\Gamma(N_+ + \alpha_1)}{\Gamma(N+\alpha_0+\alpha_1)} \prod_{j=0}^{m^*} \left(\frac{\sum\limits_{k \in R_{j-1}(c_{<j}, \mathcal{A})} (\eta^k/k!)}{(\eta^{c_j}/c_j!)} |Q_{c_j}| \right) \prod_{j=1}^{m^*} \frac{\Gamma(N_{j,0}+N_{j,1}+\alpha_0+\alpha_1)}{\Gamma(N_{j,0}+\alpha_0)\Gamma(N_{j,1}+\alpha_1)}$$

$$\frac{\lambda^{m^*}}{m^*!} \geq \frac{\Gamma(N_- + \alpha_0)\Gamma(N_+ + \alpha_1)}{\Gamma(N+\alpha_0+\alpha_1)} \prod_{j=1}^{m^*} (1 \times |Q_{c_j}|) \prod_{j=1}^{m^*} 1$$

$$\frac{\lambda^{m^*}}{m^*!} \geq \frac{\Gamma(N_- + \alpha_0)\Gamma(N_+ + \alpha_1)}{\Gamma(N+\alpha_0+\alpha_1)} \prod_{j=1}^{m^*} |Q_{c_j}|.$$

By construction we have $\prod\limits_{j=1}^{m^*} |Q_{c_j}| \geq \prod\limits_{j=1}^{m^*} b_j$, thus

$$\frac{\lambda^{m^*}}{m^*!} \geq \frac{\Gamma(N_- + \alpha_0)\Gamma(N_+ + \alpha_1)}{\Gamma(N+\alpha_0+\alpha_1)} \prod_{j=1}^{m^*} b_j.$$

We need only the first m terms of the b_j's, the rest are not needed. Note that the left hand side decreases rapidly after m exceeds λ. In addition to this inequality, there is an additional (trivial) upper limit for m, namely the value $2^P - 1$, which corresponds to a rule list that includes all of the possible rules. So the length of the optimal rule list should satisfy the following upper bound:

$$m^* \leq m_{\max} = \min \left\{ 2^P - 1, \max \left\{ m' \in \mathbb{Z}_+ : \frac{\lambda^{m'}}{m'!} \geq \frac{\Gamma(N_- + \alpha_0)\Gamma(N_+ + \alpha_1)}{\Gamma(N+\alpha_0+\alpha_1)} \prod_{j=1}^{m'} b_j \right\} \right\}.$$

□

Appendix A. Appendix: Proof of Theorem 2

For rule list $d = \{d_p, a_{p+1}, a_{p+2}, ..., a_m, a_0\}$, consider the set of data points captured by rule j,

$$S_{a_j} := \{i : \text{Captr}(i) = j\}.$$

Again recall the definition of $N_{j,0}$ as the number of points captured by rule j with label 0, and $N_{j,1}$ as the number of points captured by rule j with label 1,

$$N_{j,0} = |\{i : \text{Captr}(i) = j \text{ and } y_i = 0\}|, \quad N_{j,1} = |\{i : \text{Captr}(i) = j \text{ and } y_i = 1\}|.$$

Definition 2 *For rule j, if either $N_{j,0}$ or $N_{j,1}$ equals zero, rule j is called a* ***perfect rule*** *with respect to d.*

A perfect rule correctly classifies all observations it captures.

Lemma 1 *For rule list*

$$d = d_p, a_{p+1}, a_{p+2}, ..., a_j, ..., a_m, a_0$$

where a_j is not a perfect rule, consider a hypothetical rule list

$$d^{\text{better}} = d_p, a_{p+1}, a_{p+2}, ..., a^{j^+}, a^{j^-}, ..., a_m, a_0$$

where a^{j^+} and a^{j^-} are perfect rules with label 1's and 0's, respectively, that capture the same observations as rule j, so that $N_{j^+,1} = N_{j,1}$, $N_{j^+,0} = 0$, and $N_{j^-,1} = 0$, $N_{j^-,0} = N_{j,0}$. Then, for parameters $\alpha_0 = 1$ and $\alpha_1 = 1$,

$$\text{Likelihood}(d, \{(x_i, y_i)\}_{i=1}^n) < \text{Likelihood}(d^{\text{better}}, \{(x_i, y_i)\}_{i=1}^n).$$

(Note that a^{j^+} and a^{j^-} may not exist in practice, but we create them in theory for the purposes of this proof.)

Intuitively, Lemma 1 states that if rule j is not a perfect rule with respect to d, meaning $N_{j,0} \geq 1$ and $N_{j,1} \geq 1$, then replacing rule j with two perfect rules that capture the same data points would improve the likelihood.

Proof 1 *We compare the likelihood ratio of the rule lists before and after splitting rule j into two perfect rules. Splitting the rule will not affect the data points captured by other rules. The likelihood of a rule list is a product of likelihoods for individual rules. Thus,*

$$\begin{aligned}
&\frac{\text{Likelihood}(d^{\text{better}}, \{(x_i, y_i)\}_{i=1}^n)}{\text{Likelihood}(d, \{(x_i, y_i)\}_{i=1}^n)} \\
&= \frac{\frac{\Gamma(N_0+1)\Gamma(1)}{\Gamma(N_0+2)}\frac{\Gamma(1)\Gamma(N_1+1)}{\Gamma(N_1+2)}}{\frac{\Gamma(N_0+1)\Gamma(N_1+1)}{\Gamma(N_0+N_1+2)}} = \frac{(N_0+N_1+1)!}{(N_0+1)!(N_1+1)!} \quad \textit{(eliminated common factors)} \\
&= \frac{\binom{N_0+N_1+1}{N_0+1}}{N_1+1} \quad \left(\textit{using identity} \binom{n}{k} = \frac{n!}{k!(n-k)!}\right) \\
&= \frac{\binom{N_0+(N_1-1)+1}{N_0+1} + \binom{N_0+N_1}{N_0}}{N_1+1} \quad \left(\textit{using identity} \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}\right) \\
&= \frac{\binom{N_0+(N_1-1)+1}{N_0+1} + \binom{N_0+N_1}{N_1}}{N_1+1} \quad \left(\textit{using identity} \binom{n}{k} = \binom{n}{n-k}\right) \\
&\geq \frac{\binom{N_0+1}{N_0+1} + \binom{1+N_1}{N_1}}{N_1+1} \quad (\textit{because } N_0, N_1 \geq 1) \\
&= \frac{N_1+2}{N_1+1} \\
&> 1.
\end{aligned}$$

□

Let us discuss the next result, Lemma 2. If j and k, where $j \leq k$ are both perfect rules in d and capture data points with only label l's (where l is either 0 or 1), then replacing a_j, a_k with a single perfect rule a_{kj} that captures the same data points will improve the likelihood probability. Formally,

Lemma 2 *For rule list*

$$d = d_p, a_{p+1}, a_{p+2}, ..., a_k, ..., a_j, ...a_m, a_0 \tag{6}$$

where k and j are both perfect rules and have the same label l, consider a hypothetical rule list

$$d^{\text{consolidated}} = d_p, a_{p+1}, a_{p+2}, ..., a_{kj}, ...a_m, a_0 \tag{7}$$

where kj is a perfect rule that captures all the data points captured by k and j. Then

$$\text{Likelihood}(d, \{(x_i, y_i)\}_{i=1}^n) < \text{Likelihood}(d^{\text{consolidated}}, \{(x_i, y_i)\}_{i=1}^n). \tag{8}$$

Proof 2 *:*

$$\begin{aligned}
&\frac{\text{Likelihood}(d^{\text{consolidated}}, \{(x_i, y_i)\}_{i=1}^n)}{\text{Likelihood}(d, \{(x_i, y_i)\}_{i=1}^n)} \\
= \;& \frac{\frac{\Gamma(N_{j,l}+N_{k,l}+\alpha_l)\Gamma(\alpha_l)}{\Gamma(N_{j,l}+N_{k,l}+2\alpha_l)}}{\frac{\Gamma(N_{j,l}+\alpha_l)\Gamma(\alpha_l)}{\Gamma(N_{j,l}+2\alpha_l)}\frac{\Gamma(N_{k,l}+\alpha_l)\Gamma(\alpha_l)}{\Gamma(N_{k,l}+2\alpha_l)}} \quad \textit{(by definition)} \\
= \;& \frac{1}{\Gamma(\alpha_l)} \frac{(N_{j,l}+\alpha_l)(N_{k,l}+\alpha_l)}{(N_{j,l}+N_{k,l}+\alpha_l)} \frac{(N_{j,l}+\alpha_l+1)(N_{k,l}+\alpha_l+1)}{(N_{j,l}+N_{k,l}+\alpha_l+1)} \cdots \frac{(N_{j,l}+2\alpha_l-1)(N_{k,l}+2\alpha_l-1)}{(N_{j,l}+N_{k,l}+2\alpha_l-1)} \\
= \;& \frac{1}{\Gamma(\alpha_l)} \left[\frac{N_{j,l}N_{k,l}+\alpha_l N_{j,l}+\alpha_l N_{k,l}+\alpha_l^2}{N_{j,l}+N_{k,l}+\alpha_l}\right] \cdots \left[\frac{N_{j,l}N_{k,l}+(2\alpha_l-1)N_{j,l}+(2\alpha_l-1)N_{k,l}+(2\alpha_l-1)^2}{N_{j,l}+N_{k,l}+(2\alpha_l-1)}\right] \\
= \;& \frac{1}{\Gamma(\alpha_l)} \left[\alpha_l + \frac{N_{j,l}N_{k,l}}{N_{j,l}+N_{k,l}+\alpha_l}\right] \cdots \left[2\alpha_l - 1 + \frac{N_{j,l}N_{k,l}}{N_{j,l}+N_{k,l}+(2\alpha_l-1)}\right] \\
> \;& \frac{1}{\Gamma(\alpha_l)} [\alpha_l] [\alpha_l+1] [\alpha_l+2] \cdots [2\alpha_l-1] \\
= \;& \frac{\alpha_l}{1} \frac{\alpha_l+1}{2} \frac{\alpha_l+2}{3} \cdots \frac{\alpha_l+2\alpha_l-1}{\alpha_l} \\
\geq \;& 1.
\end{aligned}$$

□

Proof 3 *(Of Theorem 2) Combining Lemma 1 and Lemma 2, we can get an upper bound for the posterior of rule list d in terms of the first few rules in the list. Lemma 1 tells us to separate each rule hypothetically into two perfect rules. Lemma 2 tells us to combine all perfect rules from the same class into a single rule. After doing this, there are only two rules left, a perfect rule for class label 0 and a perfect rule for class label 1. We conclude that the likelihood of the rule list $d = \{d_p, a_{p+1}, a_{p+2}, ..., a_m, a_0\}$ is at most the likelihood of the rule list*

$$d^{\textit{hypothetical}} = \{d_p, a_{p_0}, a_{p_1}, a_0\},$$

where p_0 *is an imaginary perfect rule in* $d^{hypothetical}$ *capturing all remaining data points with label 0's and* p_1 *is an imaginary perfect rule in* $d^{hypothetical}$ *capturing all remaining data points with label 1's. That is:*

$$\text{Likelihood}(d, \{(x_i, y_i)\}_{i=1}^n)) \leq \text{Likelihood}(d^{hypothetical}, \{(x_i, y_i)\}_{i=1}^n)).$$

We compress notation slightly to remove explicit dependence on the data, so we write $\text{Likelihood}(d) = \text{Likelihood}(d, \{(x_i, y_i)\}_{i=1}^n)$*. Also note that the likelihood of the list can be decoupled into terms for each rule,*

$$\text{Likelihood}(d) = \prod_{j=1}^{m} \frac{\Gamma(N_{j,0} + \alpha_0)\Gamma(N_{j,1} + \alpha_1)}{N_{j,0} + N_{j,1} + \alpha_0 + \alpha_1} = \prod_{j=1}^{m} \text{Likelihood}(\textit{rule } j),$$

which means that the likelihood for rule list $d^{hypothetical}$ *can be split into likelihood for the first* p *rules and likelihood for the other rules.*

$$\begin{aligned} &\text{Likelihood}(d^{hypothetical}, \{(x_i, y_i)\}_{i=1}^n) = \\ &\quad \text{Likelihood}(d_p, \textit{data captured by rules in } d_p) \times \text{Likelihood}(a_{p_0}, \textit{data captured by } a_{p_0}) \times \\ &\quad \text{Likelihood}(a_{p_1}, \textit{data captured by } a_{p_1}) \times \text{Likelihood}(a_0, \textit{no data}). \end{aligned}$$

Next we show $\text{Posterior}(d) \leq \text{Posterior}(d^{hypothetical}) \leq \Upsilon(d_p, \{(x_i, y_i)\}_{i=1}^n)$*. We compute:*

$$\begin{aligned} \text{Posterior}(d) &= \text{Prior}(d) \times \text{Likelihood}(d) \\ &\leq \text{Prior}(d) \times \text{Likelihood}(d^{hypothetical}) \\ &= \text{Prior}(\textit{number of rules in } d) \times \text{Prior}(\textit{size of rules in } d) \times \text{Likelihood}(d^{hypothetical}) \\ &= \text{Prior}(m|\lambda) \times \text{Prior}(\textit{size of rules in } d_p) \times \text{Prior}(\textit{size of rules in } d \backslash d_p) \times \\ &\quad\; \text{Likelihood}(d_p) \times \text{Likelihood}(a_{p_0}) \times \text{Likelihood}(a_{p_1}) \times \text{Likelihood}(a_0). \end{aligned} \tag{9}$$

Let us handle each term of the expression above, starting with the term for the number of rules. The largest value of the prior occurs at the maximum of the Poisson distribution centered at λ*. That would happen if there were* λ *total rules. This could happen if* $p \leq \lambda$*. If* $p > \lambda$*, then we cannot have a rule list of size* λ *since we already have too many rules. In that case, we should not add more rules, and the maximum prior occurs when the size of the rule list is* p*. That is,*

$$\text{Prior}(m|\lambda) \leq \frac{\lambda^{\max(p,\lambda)}/(\max(p,\lambda))!}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}.$$

The second term in (9) *is an equality,*

$$\text{Prior}(\textit{size of rules in } d_p) = \left(\prod_{j=1}^{p} p(c_j | c_{<j}, \mathcal{A}, \eta) \frac{1}{|Q_{c_j}|} \right).$$

The third term in (9) *is trivially bounded by 1. The fourth term* $\text{Likelihood}(d_p)$ *can be calculated from the data as usual, simplifying with* $\alpha_0 = \alpha_1 = 1$*:*

$$\text{Likelihood}(d_p) = \prod_{j=1}^{p} \frac{\Gamma(N_{j,0} + \alpha_0)\Gamma(N_{j,1} + \alpha_1)}{\Gamma(N_{j,0} + N_{j,1} + \alpha_0 + \alpha_1)} = \prod_{j=1}^{p} \frac{\Gamma(N_{j,0} + 1)\Gamma(N_{j,1} + 1)}{\Gamma(N_{j,0} + N_{j,1} + 2)}$$

For the terms for hypothetical rules a_{p_0} and $a_{p,1}$ we compute them as if those rules were real rules:

$$\begin{aligned}
\text{Likelihood}(a_{p,0}) &= \frac{\Gamma(1+N_0-\sum_{j=1}^{p}N_{j,0})}{\Gamma(2+N_0-\sum_{j=1}^{p}N_{j,0})}\\
\text{Likelihood}(a_{p,1}) &= \frac{\Gamma(1+N_1-\sum_{j=1}^{p}N_{j,1})}{\Gamma(2+N_1-\sum_{j=1}^{p}N_{j,1})}.
\end{aligned}$$

The last term of (9) *will be trivially upper bounded by 1. Multiplying all of these terms together to form an upper bound, we have precisely the definition of* $\Upsilon(d_p, \{(x_i, y_i)\}_{i=1}^n)$*. Thus,*

$$\text{Posterior}(d) \le \Upsilon(d_p, \{(x_i, y_i)\}_{i=1}^n).$$

By the assumption of Theorem 2, we know that for our rule list d,

$$\text{Posterior}(d) \quad \le \quad \Upsilon(d_p) < v_t^* = \max_{t' \le t} \text{Posterior}(d^{t'}, \{(x_i, y_i)\}_{i=1}^n) \le \max_{d'} \text{Posterior}(d'),$$

and more simply stated,

$$\text{Posterior}(d) \quad < \quad \max_{d'} \text{Posterior}(d').$$

Thus, there is no possible way that our current rule list d could be within argmax$_{d'}$ Posterior(d')*.*
□