Scalable Bayesian Rule Lists

Hongyu Yang [1] Cynthia Rudin [2] Margo Seltzer [3]

Abstract

We present an algorithm for building probabilistic 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 aim to 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, with practical running times. The predictions in each leaf are probabilistic.

1. Introduction

Our goal is to build a competitor for decision tree and rule learning 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 composed 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 (see Letham et al., 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 rule in the rule list contains (“captures”) a number of observations that is bounded below. Because of this approximation, the set of conditions defining each leaf is a frequent pattern. This means the antecedents within the rule list are all 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 antecedents and their order to create the rule list.

(iii) **High performance language libraries** to achieve computational efficiency. Optimization over rule lists is done through repeated low level computations. At every iteration, we make a change to the rule list and need to evaluate the new rule list on the data. High performance calculations (novel to this problem) 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

[1]Massachusetts Institute of Technology, Cambridge, Massachusetts, USA [2]Duke University, Durham, North Carolina, USA [3]Harvard University, Cambridge, Massachusetts, USA. Correspondence to: Hongyu Yang <hongyuy@mit.edu>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

Table 1. Rule list for the Mushroom dataset from the UCI repository (data available from Bache & Lichman, 2013). PP: the probability that the label is positive (the mushroom is edible).

	RULE-LIST	PP
IF	BRUISES=NO,ODOR=NOT-IN-(NONE,FOUL)	0.001
ELSE IF	ODOR=FOUL,GILL-ATTACHMENT=FREE,	0.001
ELSE IF	GILL-SIZE=BROAD,RING-NUMBER=ONE,	0.999
ELSE IF	STALK-ROOT=UNKNOWN,STALK-SURFACE-ABOVE-RING=SMOOTH,	0.996
ELSE IF	STALK-ROOT=UNKNOWN,RING-NUMBER=ONE,	0.039
ELSE IF	BRUISES=NO,VEIL-COLOR=WHITE,	0.995
ELSE IF	STALK-SHAPE=TAPERING,RING-NUMBER=ONE,	0.986
ELSE IF	HABITAT=PATHS,	0.958
ELSE	(DEFAULT RULE)	0.001

rules and providing bounds on the optimal solution. These are provided in two theorems in this paper.

Through a series of controlled experiments, we show that SBRL is over two orders of magnitude faster than the previous best code for this problem.

For example, Table 1 presents 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 edible. It was created in about 9 seconds on a laptop and achieves perfect out-of-sample accuracy.

2. Review of Bayesian Rule Lists

Scalable Bayesian Rule Lists maximizes 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 rule list has the following form:

if x obeys a_1 **then** $y \sim \text{Binom}(\theta_1)$,
$\theta_1 \sim \text{Beta}(\alpha + \mathbf{N}_1)$
else if x obeys a_2 **then** $y \sim \text{Binom}(\theta_2)$,
$\theta_2 \sim \text{Beta}(\alpha + \mathbf{N}_2)$
$\vdots$
else if x obeys a_m **then** $y \sim \text{Binom}(\theta_m)$,
$\theta_m \sim \text{Beta}(\alpha + \mathbf{N}_m)$
else $y \sim \text{Binom}(\theta_0)$, $\theta_0 \sim \text{Beta}(\alpha + \mathbf{N}_0)$.

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 $\alpha = [\alpha_\mathbf{1}, \alpha_\mathbf{0}]$ 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 $\theta_j|\alpha \sim \text{Beta}(\alpha)$. 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 α 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 $j =$ argmin j' such that $a_{j'}(x_i) =$ 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 those 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. It is an optimization over subsets of rules and their permutations.

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, $d = (m, \{a_j, \theta_j\}_{j=0}^m)$. 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.

We next discuss the prior. It has three terms, one governing the number of rules m in the list, one governing the size c_j of each antecedent j, and one governing the choice of antecedent condition a_j of rule j given its size. Specifically, c_j is the cardinality of antecedent a_j, also written $|a_j|$, the number of conjunctive clauses in antecedent a_j. E.g, if a is 'x_1=green' and 'x_2<50', this has cardinality 2. Notation $a_{<j}$ includes the antecedents before j in the rule list, if there are any, *e.g.* $a_{<4} = \{a_1, a_2, a_3\}$. $c_{<j}$ 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:

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

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 $R_{j-1}(c_{<j},\mathcal{A})$ 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 have a_j selected from a uniform distribution over antecedents in $\mathcal{A}$ of size c_j, excluding those in $a_{<j}$.

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

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 optimize 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 to be 1 to avoid favoring one class label over another.

Given the prior parameters λ, η, and α, 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.

3. Representation

We use an MCMC scheme: at each time t, we choose a neighboring rule list at random by adding, removing, or swapping rules, starting with the basic algorithm of Letham et al. (2015) as a starting point. However, to optimize performance we use a more efficient rule list representation that is amenable to fast computation.

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. The naïve 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 efficient in terms of both memory and computation. 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 a 128 KB vector (since a byte consists of 8 bits), which fits comfortably in most L2 caches.

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 (unique) rule in the set captures which observation. 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 n rules. Imagine that we wish to delete rule k from the set. The naïve 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%.

A fast algebra for computational 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. We have custom implementations (discussed in the full version of this work, Yang et al., 2017) for the following: (i) *Remove rule k* uses boolean operations on bit vectors to redistribute the observations captured by rule k to the rules below it in the list. (ii) *Insert rule k* shifts the rules below k down one position, determines which observations are captured by the new rule, and removes those observations from the rules below it. (iii) *Swap consecutive rules* updates only which observations were captured for the two swapped rules. (iv) *Generalized swap* subroutine updates only observations captured for all rules between the two rules to be swapped. All operations use only bit vector computations.

Ablation study: Having transformed expensive set operations into bit vector operations, we can now leverage both hardware vector instructions and optimized software libraries. We investigated four alternative implementations, each improving efficiency from the previous one. (i) First, we have the original python implementation here for comparison. (ii) Next, we retained our python implementation but converted from set operations to bit operations. (iii) Then, we used the python gmpy library to perform the bit operations. (iv) Finally, we moved the implementation from Python to C, represented the bit vectors as multi-precision integers, used the GMP library, which is faster on

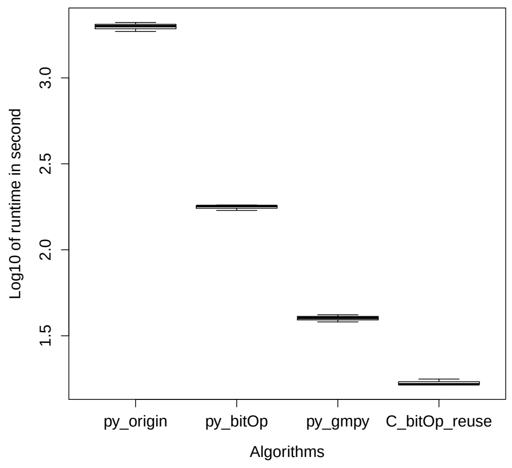

Figure 1. Boxplots of $\log_{10}$ runtime among different implementations. From the original python code, the final code is over two orders of magnitude faster.

large data sets, and implemented the algebra for computational reuse outlined above. 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 four 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 $\log_{10}$ of the run time over the different folds, which are shown in Figure 1. The final code is over two orders of magnitude faster than the original optimized python code (that of Letham et al., 2015).

4. Bounds

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

4.1. 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 antecedents and binary features (e.g., a_j is true if female), so the total number of possible

Algorithm 1 Calculating b_j's

```
Initialization: index=0, $b_0 = 1$
for $c = 0$ to $\lfloor \frac{P}{2} \rfloor$ do
  for $j = \binom{P}{c}$ downto $1$ do
    index = index + 1
    $b_{\text{index}} = j$
  end for
  if $c + c \neq p$ then
    for $j = \binom{P}{P-c}$ downto $1$ do
      index = index + 1
      $b_{\text{index}} = j$
    end for
  end if
end for
```

antecedents of each size can be calculated directly. When creating the upper bound, within the proof, we hypothetically exhaust antecedents from each size category in turn, starting with the smallest sizes. We discuss this further below.

Let $|Q_c|$ be the number of antecedents 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_0, b_1, b_2, etc. represents the sequence of sizes of rules that would provide the smallest possible $|Q_c|$'s. Intuitively, the sequence of b's arises when we deplete the antecedents of size 1, then deplete all of size 2, etc. The number of ways to do this is given by the b_{index} values, computed as Algorithm 1, where P is the number of features, and $\mathbf{b} = \{b_0, b_1, ...b_{2^P-1}\}$ is a vector of length 2^P. We will use $\mathbf{b}$ 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*

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

With parameters $\alpha_0 = 1$ and $\alpha_1 = 1$, this reduces to:

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

The proof is in the longer version of this paper (Yang et al., 2017). Theorem 1 tends to significantly reduce the size of the space. Without this bound, it is possible that the search would consider extremely long lists, without knowing that they are provably non-optimal.

4.2. Prefix Bound

We next provide a bound that eliminates certain regions of the rule space from consideration. Consider a rule list beginning with antecedents $a_1, .., a_p$. 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 that start 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 antecedents 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 us consider a rule list with antecedents $d = (a_1, a_2, ...a_m, a_0)$. 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 $a_1, a_2, ..., a_p$ are the same as the first p antecedents in d. 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:

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

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,

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

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

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

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. The proofs are lengthy and contained in the longer version of this work (Yang et al., 2017).

5. Experiments

We provide a comparison of algorithms along three dimensions: solution quality (AUC - area under the ROC curve), sparsity, and scalability. As baselines, we chose popular algorithms to represent the sets of uninterpretable methods and the set of "interpretable" methods. To represent uninterpretable methods, we chose logistic regression, SVM RBF, random forests (RF), and boosted decision trees (ADA). To represent the class of "interpretable" algorithms, we chose CART, C4.5, RIPPER (Cohen, 1995), CBA (Liu et al., 1998), and CMAR (Li et al., 2001). Other works (see Letham et al., 2015; Wang & Rudin, 2015a) have accuracy/interpretability comparisons to Bayesian Rule Lists and Falling Rule Lists, so our main effort here will be to study the scalability component. Implementation details are in the full version (Yang et al., 2017).

We benchmark using publicly available datasets (see Bache & Lichman, 2013) that have interpretable features: the Tic Tac Toe dataset, where the goal is to determine whether the "X" player wins; the Adult dataset, where we aim to predict whether an individual makes over \$50K peryear; the Mushroom dataset, where the goal is to predict whether a mushroom is edible; the Nursery dataset, where the goal is to predict whether a child's application to nursery school will be in either the "very recommended" or "special priority" categories; and the Telco customer churn dataset (see WatsonAnalytics, 2015), where the goal is to predict whether a customer will leave the service provider.

Evaluations of prediction quality, sparsity, and timing were done using 10-fold cross validation. 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 ran 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 of the 20 chains of SBRL, which we ran in series on a laptop. 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

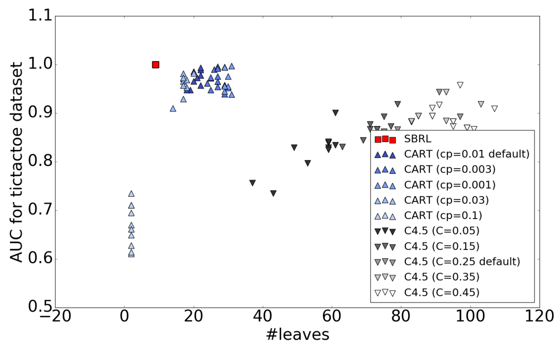

Figure 2. Scatter plot of AUC against the number of leaves (sparsity) for Tic Tac Toe. There is a triangle for each of 10 folds, for several settings of CART and C4.5's parameters.

Table 2. Run Time on Adult dataset

Run Time	LR	SVM	RF	ADA	CART	C4.5	RIPPER	CBA	CMAR	SBRL
Mean	1.353	238.5	23.53	45.71	0.809	0.512	1005.9	2.557	3.000	17.97
Median	1.406	239.9	23.56	43.62	0.813	0.513	1005.4	2.520	2.920	18.01
STD	0.203	5.693	0.133	4.672	0.022	0.011	100.14	0.174	0.191	0.171

than the best rule list we had found so far. If not, the rule was replaced until the condition was satisfied.

Tic Tac Toe: 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. Figure 2 shows a scatter plot of AUC vs. number of leaves (sparsity), where each triangular marker represents an evaluation of one algorithm, on one fold, with one parameter setting. 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 SBRL. SBRL's run time was on average 0.759 (± .02) seconds.

Adult: For the Adult dataset, results are in Figure 3, Figure 4 and Table 2. 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 information. 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 about 18 seconds, where 14 seconds were MCMC iterations. If the chains were computed in parallel rather than in series, it would speed up computation further.

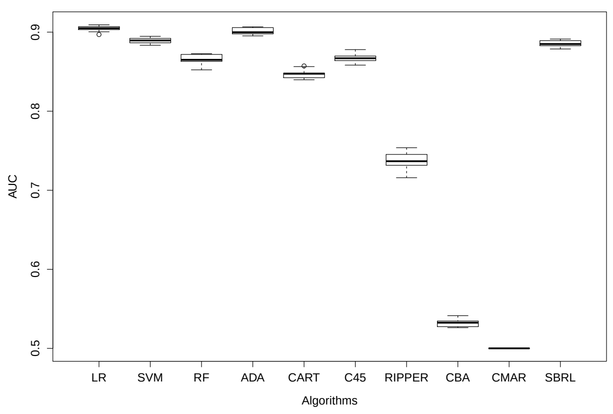

Figure 3. Comparison of AUC among different methods on the Adult dataset.

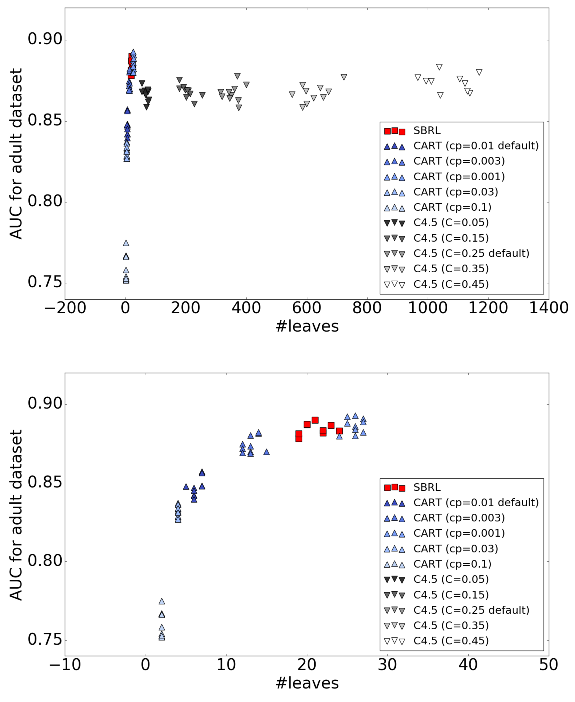

Figure 4. Scatter plots of AUC against the number of leaves for the Adult dataset. 10 folds are included, along with results from several different settings of CART and C4.5's parameters. The bottom is a zoomed-in version of the top.

Mushroom: Table 1 contains an SBRL rule list for Mushroom; other results are in Yang et al. (2017). SBRL attains perfect test accuracy on this dataset.

Nursery: Results from the Nursery dataset are shown in Figure 5. 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.

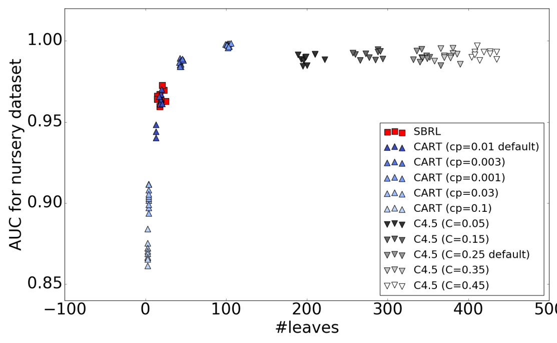

Figure 5. Scatter plot of AUC against the number of leaves (sparsity) for Nursery. 10 folds are included, along with results from several different settings of CART and C4.5's parameters.

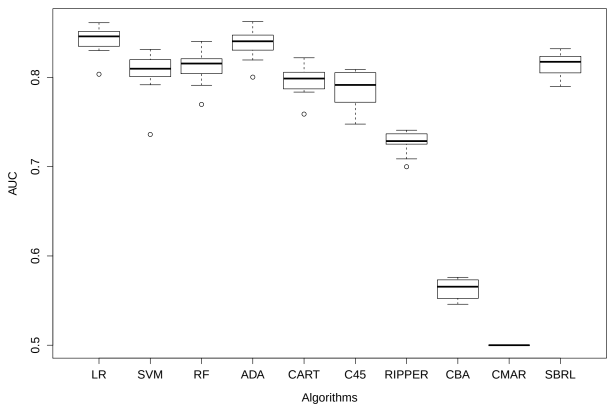

Figure 6. Comparison of AUC of ROC among different methods on Telco.

Telco: Figure 6, Figure 7 and Table 3 show the results for the Telco dataset, which contains 7043 observations and 18 features. Similar observations hold for this dataset. The model from one of the ten folds is provided in Table 4.

6. Scalability

We used the USCensus1990 data (see Bache & Lichman, 2013) to test the scalability of SBRL on large datasets. We used 1,000,000 observations and set SBRL's parameter to extract ≈1000 rules as problem (A) and about 50,000 observations with 50,000 rules as problem (B). The runtime comparison with CART is shown in Table 5. For problem (A), the run times are similar; for (B); SBRL is slower (2.5 hours), which is not prohibitive for important problems. One can see why CART does not perform as well in high dimensions, as it often spends less time on harder problems than on easier ones; details are in (Yang et al.,

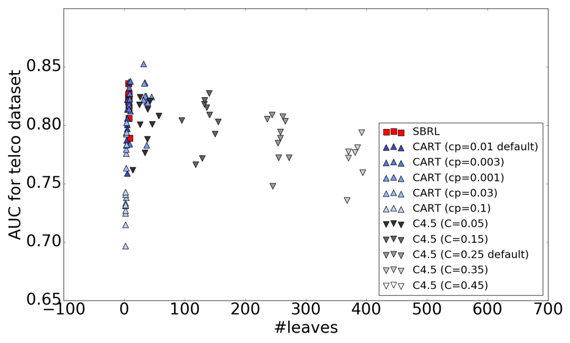

Figure 7. Scatter plot of AUC against the number of leaves (sparsity) for Telco. 10 folds are included, along with results from several different settings of CART and C4.5's parameters.

Table 3. Run Time on Telco dataset

Run Time	LR	SVM	RF	ADA	CART	C4.5	RIPPER	CBA	CMAR	SBRL
Mean	0.267	7.468	3.703	7.839	0.168	0.250	37.14	8.028	1.679	5.239
Median	0.272	7.550	3.695	8.726	0.168	0.252	37.63	8.050	1.705	5.271
STD	0.009	0.207	0.183	0.111	0.008	0.017	3.202	0.400	0.161	0.149

2017).

7. Related Works and Discussion

Rule lists are a type of one-sided decision tree, and any decision tree can be written as a rule list by enumerating the leaves. Thus SBRL is a competitor for CART (Breiman et al., 1984). CART is currently still popular in industry. CART and other decision tree methods (also decision list methods and associative classification methods) form trees from the top down using greedy splitting and greedy pruning (see, e.g., Quinlan, 1983; Clark & Niblett, 1989; Cendrowska, 1987; Rivest, 1987; Quinlan, 1993; Liu et al., 1998; Li et al., 2001; Yin & Han, 2003; Marchand & Sokolova, 2005; Vanhoof & Depaire, 2010; Rudin et al., 2013). Since our work does not use greedy splitting and pruning, it is closer to Bayesian tree models (Dension et al., 1998; Chipman et al., 2002; 2010), which are built greedily from the top down, but then the trees change according to an MCMC scheme, which allows for more exploration of the search space. However even with MCMC, the chains tend to center on local optima. It may be possible to use our techniques to build trees, where one would mine rules and create a globally optimal tree.

There are a series of works from the mid-1990's onwards on finding optimal decision trees using dynamic programming and search techniques (e.g., Bennett & Blue, 1996; Auer et al., 1995; Dobkin et al., 1996; Boros et al., 2000; Garofalakis et al., 2000; Farhangfar et al., 2008), mainly working with fixed depth trees. The number of trees of a

Table 4. Example of rule list for Telco-Customer-Churn dataset fold 1 (CV1). PP: positive probability. TA: test accuracy.

	Rule-list	PP	TA
IF	(Contract=One_year &Streaming-Movies=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 5. Run time (in seconds) and AUC of SBRL on USCensus1990 dataset. A: 1 million data points and 1 thousand rules; B: 50 thousand data points and 50 thousand rules.

Run Time(s)	SBRL	CART
(A)	2700	2000
(B)	9000	84

AUC	SBRL	CART
(A)	0.940	0.886
(B)	0.925	0.906

fixed depth is much larger than the number of rule lists of a fixed depth and are therefore more difficult to optimize. Nijssen & Fromont (2010) use pre-mined rules to form trees, but in a different way than our method. There, the user premines all possible *leaves*, enumerating all conditions leading to that leaf. (By contrast, in associative classification, we mine only small conjunctions, and their ordered combination creates leaves.) As as result, Nijssen & Fromont (2010) warn about issues related to running out of memory.

An extension of this work (CORELS - Angelino et al., 2017) does not provide probabilistic predictions, but is able to provide a certificate of optimality to a globally optimal rule list. This indicates that SBRL is probably also achieving optimality; however, because SBRL is probabilistic, the proof of optimality is much more difficult. To clarify: finding the optimal solution for both methods should be approximately equally difficult, but proving optimality for SBRL is much more difficult. However, there is a clear practical benefit to having probabilistic predictions like those of SBRL. One can post-process CORELS to have probabilistic predictions by computing $P(Y = 1|x \in \text{leaf})$ for each leaf, but this is not the same as optimizing likelihood and obtaining these probabilities directly like SBRL.

There are several subfields that produce disjunctive normal form (DNF) classifiers rather than rule lists, including rule learning/induction, and associative classification, which stemmed possibly from work in the 1960s (Michalski, 1969), and throughout the 1980's and 90's (Cendrowska, 1987; Clark & Niblett, 1989; Cohen, 1995). The vast majority of these techniques are not probabilistic, and aim for covering the positive class without covering the negative class. Rijnbeek & Kors (2010) aim to produce globally optimal DNF models. There is recent work on probabilistic DNF's that is similar to SBRL (Wang et al., 2016; 2017).

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

SBRL aims to produce interpretable models. Interpretability has been a fundamental topic in artificial intelligence for a long time (see Rüping, 2006; Bratko, 1997; Dawes, 1979; Vellido et al., 2012; Giraud-Carrier, 1998; Holte, 1993; Shmueli, 2010; Huysmans et al., 2011; Freitas, 2014). Because the rule lists created by our method are designed to be interpretable, one would probably not want to boost them using AdaBoost to form more complicated models. This contrasts with, for instance, Friedman & Popescu (2008), who linearly combine pre-mined rules.

Rule lists and their variants are currently being used for text processing (King et al., 2017), discovering treatment regimes (Zhang et al., 2015; Lakkaraju & Rudin, 2017; Wang & Rudin, 2015b), and creating medical risk assessments (Letham et al., 2015), among other applications.

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 such as 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 does not use 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 or interpretable trees, 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.

Code

Code for SBRL is available at the following link: `https://github.com/Hongyuy/sbrlmod`
Link to R package SBRL on CRAN: `https://cran.r-project.org/web/packages/sbrl/index.html`

Acknowledgement

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

References

Angelino, Elaine, Larus-Stone, Nicholas, Alabi, Daniel, Seltzer, Margo, and Rudin, Cynthia. Certifiably optimal rule lists for categorical data. In *Proceedings of the 23rd ACM SIGKDD Conference of Knowledge, Discovery, and Data Mining (KDD)*, 2017.

Auer, Peter, Holte, Robert C., and Maass, Wolfgang. Theory and applications of agnostic PAC-learning with small decision trees. pp. 21–29. Morgan Kaufmann, 1995.

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

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

Boros, Endre, Hammer, Peter L., Ibaraki, Toshihide, Kogan, Alexander, Mayoraz, Eddy, and Muchnik, Ilya. An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):292–306, 2000.

Bratko, I. Machine learning: between accuracy and interpretability. In Della Riccia, Giacomo, Lenz, Hans-Joachim, and Kruse, Rudolf (eds.), *Learning, Networks and Statistics*, volume 382 of *International Centre for Mechanical Sciences*, pp. 163–177. Springer Vienna, 1997.

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

Cendrowska, J. PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4): 349–370, 1987.

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

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

Clark, Peter and Niblett, Tim. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

Cohen, William W. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123. Morgan Kaufmann, 1995.

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

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

Dobkin, David, Fulton, Truxton, Gunopulos, Dimitrios, Kasif, Simon, and Salzberg, Steven. Induction of shallow decision trees. Citeseer, 1996.

Farhangfar, Alireza, Greiner, Russell, and Zinkevich, Martin. 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*, 2008.

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

Friedman, Jerome H. and Popescu, Bogdan E. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008.

Garofalakis, Minos, Hyun, Dongjoon, Rastogi, Rajeev, and Shim, Kyuseok. Efficient algorithms for constructing decision trees with constraints. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 335–339, New York, NY, USA, 2000. ACM.

Giraud-Carrier, Christophe. 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, 1998.

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

Huysmans, Johan, Dejaeger, Karel, Mues, Christophe, Vanthienen, Jan, and Baesens, Bart. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

King, Gary, Lam, Patrick, and Roberts, Margaret. Computer-assisted keyword and document set discovery from unstructured text. *American Journal of Political Science*, 2017.

Lakkaraju, Himabindu and Rudin, Cynthia. Learning cost effective and interpretable treatment regimes in the form of rule lists. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, 2017.

Langley, P. Crafting papers on machine learning. In Langley, Pat (ed.), *Proceedings of the 17th International Conference on Machine Learning ICML*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

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

Li, Wenmin, Han, Jiawei, and Pei, Jian. CMAR: accurate and efficient classification based on multiple class-association rules. In *IEEE International Conference on Data Mining*, pp. 369–376, 2001.

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

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

Michalski, R. S. On the quasi-optimal solution of the general covering problem. In *Proceedings of the V International Symposium on Information Processing (FCIP 69)*, pp. 125–128, 1969.

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

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

Quinlan, J. Ross. *Learning Efficient Classification Procedures and Their Application to Chess End Games*, pp. 463–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

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

Rijnbeek, Peter R. and Kors, Jan A. Finding a short and accurate decision rule in disjunctive normal form by exhaustive search. *Machine Learning*, 80(1):33–62, 2010.

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

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

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

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

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

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

Wang, Fulton and Rudin, Cynthia. Falling rule lists. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, 2015a.

Wang, Fulton and Rudin, Cynthia. Causal falling rule lists. *CoRR*, abs/1510.05189, 2015b. URL http://arxiv.org/abs/1510.05189.

Wang, Tong, Rudin, Cynthia, Doshi, Finale, Liu, Yimin, Klampfl, Erica, and MacNeille, Perry. Bayesian or's of and's for interpretable classification. In *SIAM International Conference on Data Mining (ICDM)*, 2016.

Wang, Tong, Rudin, Cynthia, Doshi, Finale, Liu, Yimin, Klampfl, Erica, and MacNeille, Perry. A Bayesian framework for learning rule sets for interpretable classification. *Journal of Machine Learning Research*, 2017. Accepted.

WatsonAnalytics, IBM, 2015. https://community.watsonanalytics.com/wp-content/uploads/2015/03/WA_Fn-UseC_-Telco-Customer-Churn.csv.

Yang, Hongyu, Rudin, Cynthia, and Seltzer, Margo. Scalable Bayesian rule lists for interpretable machine learning. *CoRR*, abs/1602.08610, 2017. URL http://arxiv.org/abs/1602.08610.

Yin, Xiaoxin and Han, Jiawei. CPAR: classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining (ICDM)*, pp. 331–335, 2003.

Zhang, Yichi, Laber, Eric B., Tsiatis, Anastasios, and Davidian, Marie. Using decision lists to construct interpretable and parsimonious treatment regimes. *Biometrics*, 71(4):895–904, 2015.