Random Forest Rule Extraction

William Jardee willjardee@gmail.com
Physics
Montana State University
Bozeman, MT 59715, USA

Lin Shi linshi1768@gmail.com
Computer Science
Montana State University
Bozeman, MT 59715, USA

Editor: N/A

Abstract

1. Introduction

2. Related Works

2.1 Decision trees and random forests

2.2 Tree extraction from random forest

2.3 Rule representation of neural networks

3. Rule Extraction from Random Forests

3.1 Co-variance matrix for forest clauses

Each path in a decision tree can be extracted as a logical rule. Consider a path in a binary decision tree that follows

$$\neg A \wedge \neg B \wedge C \rightarrow W_0\,,$$

using material implication and De Morgan's law this rule can be written as

$$\neg(\neg A) \vee \neg(\neg B) \vee \neg(C) \vee W_0\,. \tag{1}$$

Notice that we have decided to not cancel out the negations, this is will be useful for later as they will cancel out. Each path of each tree in the random forest can be extracted as one of these rules. We propose that if the problem can be explained with high order logical rules these rules may be embedded in these tree paths, and consequently also in these extracted rules. If two clauses show up together often we propose that they probably come from the same logical rule.

For simplicity, all of the features are assumed to be binary. Let us construct a $(2n+c)\times(2n+c)$ matrix of all zeros, where n is the number of features and c is the number of classes. Each of the rows and columns will correspond to a possible logical decision. For the example in equation 1 the first $2n$ rows would correspond to $[\neg(A), \neg(B), \neg(C), A, B, C]$ and the last c would correspond to the possible classes. Each time a pair of clauses show up together in a rule, 1 will be added to the corresponding cell. This describes a co-variance matrix.

To illustrate this, see the rule matrix, Δ, for the running example:

$$\Delta = \begin{array}{c|cccccccc|} & A & B & C & \bar{A} & \bar{B} & \bar{C} & W_0 & W_1 \\ A & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ B & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ C & \cdot & \cdot & 1 & 1 & 1 & \cdot & 1 & \cdot \\ \bar{A} & \cdot & \cdot & 1 & 1 & 1 & \cdot & 1 & \cdot \\ \bar{B} & \cdot & \cdot & 1 & 1 & 1 & \cdot & 1 & \cdot \\ \bar{C} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ W_0 & \cdot & \cdot & 1 & 1 & 1 & \cdot & 1 & \cdot \\ W_1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array},$$

where $\bar{A} \equiv \neg A$ and zeros have been replaced with $\cdot$ for readability. Adding up the Δ for each path in the forest gives a complete co-variance matrix. Similar to in Principle Component Analysis (PCA), dimensionality reduction can be done by using eigen decomposition.

3.2 Rule set extraction from co-variance matrix

Given the eigen decomposition of the matrix, represented in the form of the set of eigenvalues, λ, and it's corresponding eigenvector, $\vec{v}$, a reduced representation of the co-variance matrix can be extracted. Eigenvectors with larger λ have more importance to the matrix, as the linear transformation favors that direction. Consequently, the k most important vectors can be chosen by picking the eigenvectors that correspond to the k largest eigenvalues. To pick out only the important characteristics, the first $2n$ values of the eigenvector and the last c should be pruned individually. This is to decrease the amount of input noise from the features and pick out the important classes of the rule this vector should describe.

Because the distribution of the component values is not known, sophisticated models shouldn't be used here; i.e., a one standard deviation cut. For this paper we take the 4th quartile of the features. Future work should look into how to include the weights into rule expression, but we choose to set all rule values to 1 for simplicity. The feature portion of the eigenvector can be explained by the function

$$f_f(x_i) = \begin{cases} 1 & \text{if } x_i > 75\% \text{ of vector} \\ 0 & \text{Otherwise} \end{cases}. \tag{2}$$

For the classes portion of the eigenvector, values are kept if their contribution is greater than random. For a normalized vector with equal weighting on all components, each has a value of $1/\sqrt{c}$. For all the values larger than random the value is kept, otherwise it is set to zero. This is explained with the function

$$f_c(x_i) = \begin{cases} 1 & \text{if } x_i > 1/\sqrt{c} \\ 0 & \text{Otherwise} \end{cases}. \tag{3}$$

The resulting vector is normalized and each component squared to provide a probability measure for each class. In the current state of the project, this quantitative value is not used, but it could be either reported with the rules to educate how likely each class outcome is, or used in a scoring metric.

There is no guarantee that the resulting rules fully cover the class-space, which means that the full problem cannot be explained by the extracted rules so far. To account for this, the remaining eigenvectors should be searched for rules that cover missing classes. This is done with a greedy approach that adds the eigenvector with the largest eigenvalue to cover the missing classes. The preferred number of times a class is covered is not clear, as only one rule that describes the class may be insufficient. So, the number of times each class must show up, k^*, should be tuned. If k^* is zero, then there is no enforcement that each class is covered.

For an example, take the extracted eigenvector

$$\begin{array}{rcccccccccc} & A & B & C & \bar{A} & \bar{B} & \bar{C} & W_0 & W_1 & W_2 & W_3 \\ \vec{v} = & (6 & 5 & 4 & 3 & 2 & 1 & 0.3 & 0.1 & 0.3 & 0) \end{array}$$

where the number of classes has been increased to four from the previous example. The fourth quartile of the features will be kept (rounding down); i.e., $(1, 1, 0, 0, 0, 0)$ and the values larger than 0.125 for the classes will be kept; i.e., $(1, 0, 1, 0)$. But, recall that there is a special procedure for the class vectors, so the class vector is $(0.5, 0, 0.5, 0)$. Putting this together extracts the rule vector

$$\begin{array}{rcccccccccc} & A & B & C & \bar{A} & \bar{B} & \bar{C} & W_0 & W_1 & W_2 & W_3 \\ \vec{v}\,' = & (1 & 1 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0) \end{array} .$$

The extraction of logical rules from the rule vectors will mirror the way they were encoded. That is by a conjunction of OR statements, with the features gaining a negation. The resulting form can be considered a horn statement, where the only positive clause is the union of classes. The eigenvalue information and ratio of classes can be extracted as well.

Continuing the running example, and assigning the arbitrary eigenvalue of $\lambda = 0.8$, the extracted rule would look like

$$\begin{aligned} \neg(A) \vee \neg(B) \vee W_0 \vee W_1 \qquad & \lambda = 0.8 \qquad [0.5, 0.5]\,, \\ A \wedge B \rightarrow (W_0 \vee W_1) \qquad & \lambda^2 = 0.64 \qquad [0.5, 0.5]\,. \end{aligned}$$

The square of an eigenvalue, when all the eigenvalues are normalized, corresponds to the percentage of the co-variance matrix that is described by the corresponding eigenvector. Thus, it would make sense to report λ^2 with the rules as a measure of relative importance. It should be clear now why k^* was introduced. many rules will probably imply the existence of a possible set of classes. k^* greater than one allows more descriptive rule sets.

3.3 Time complexity

The theoretical time complexity of the algorithm will be given based on the three section of the algorithm, as can be seen in algorithm 1. The time required to build the forest will not be considered in this derivation. However, from running the empirical tests, the realistic time to run the algorithm is not vanishingly small, but the computational cost of the whole process is driven by the number of trees generated during the random forest generation.

Take the following definition of terms

$$\begin{aligned} D &\equiv \text{ Number of trees in the forest} & n &\equiv \text{ Number of features} \\ l &\equiv \text{ Maximum tree length} & c &\equiv \text{ Number of classes}\,. \end{aligned}$$

The number of paths in a full tree will be on the order of $l!$, so the time complexity of creating the complete rule set will be $\mathcal{O}(l!D)$. For the construction of the co-variance matrix each possible clause of each rule must be checked, so that is $\mathcal{O}(l!D(2n + c)^2)$, assuming that the number of features is larger than the number of classes, $\mathcal{O}(n^2 l!D)$. Finally, the rule extraction will be driven by the eigenvalue decomposition of the data, which is roughly $\mathcal{O}(n^{2.3})$ (Strang, 2006). If the number of trees is assumed to be much larger than the number of features, then the time complexity of the whole algorithm becomes $\mathcal{O}(n^2 l!D)$.

4. Experimentation

4.1 Proposed performance metric

To measure the performance of the derived rule set with a test set, a new metric was developed. The motivation of this metric is that for a given input vector, $\mathbf{X}$, and related class, y, the importance of

Algorithm 1 RFRE (*Random Forest Rule Extraction*)

```
# Creating random forest rule-set
$rf$ ← RandomForestGeneration
$extractedRules$ ← [ ]
for $t$ in $rf$ do
    $treeRules$ ← [ ]
    for $rule$ in $t$ do
        $treeRules$ ← $treeRules$ + $rule$
    end for
    $extractedRules$ ← $extractedRules$ + $treeRules$
end for
# Creating co-variance matrix for the rule-set
$n$ ← (number of features × 2) + (number of classes)
$Map$ ← $n \times n$ matrix of zeros
for $rule$ in $extractedRules$ do
    if feature $i$ and feature $j$ in $rule$ then
        $Map_{ij} \leftarrow Map_{ij} + 1$
        $Map_{ji} \leftarrow Map_{ji} + 1$
    end if
end for
# Rule extraction from co-variance matrix
$w$, $v$ ← Eigenvalues of $Map$, Eigenvectors of $Map$
$finalRules$ ← {}
for $vec$ in $v$ do
    newRule ← rule_creation($vec$)
    if newRule meets add criteria then
        $finalRules$ ← $finalRules$ + newRule
    end if
end for
return $finalRules$
```

rules that the **X** fit need to be weighted heavier than those it does not. The measurement of how well **X** matches up is done through a dot product in Cartesian space (however, there is no reason that a more complicated metric or kernel function could not be applied to the inner-product). To test the negative nodes, i.e. $\bar{A}$, **X** can be subtracted from the unary vector, notated as $\bar{\mathbf{X}}$. This gives a vector of all the values the vector is *not*. The inner product of $\bar{\mathbf{X}}$ and $\bar{A}$ will represent how well the vector lines up with the negative clauses of the rule. Since the positive examples are more specific, the weighting of them should be larger. To account for this, the inner product of the negative clauses is divided by $n-1$, where n is still the number of features. As this is specifically in the context of a dot product in Cartesian space, if a metric or kernel is being introduced this regularization should be changed.

To ensure that these two products are on the same magnitude, all the vectors must be normalized before taking the inner product. By the Schwarz' Inequality, for two vectors v and u the inner product of the two, $<\,,\,>$, must satisfy

$$\langle\, u, v \,\rangle \;\leq\; ||u|| \cdot ||v||$$

(Strang, 2006). If all the vectors are normalized before taking any inner products then $\langle\, \mathbf{X}, A \,\rangle \leq 1$ and $\langle\, \bar{\mathbf{X}}, \bar{A} \,\rangle \leq 1$. So, for a given rule, the weight can be calculated with

$$\alpha_i = \langle\, \mathbf{X}, A \,\rangle + \langle\, \bar{\mathbf{X}}, \bar{A} \,\rangle/(n-1)\,,$$

where $\alpha_i \in [0, 1 + 1/(n-1)]$.

If the class y is in the rule's conclusion, W, then the two can be thought of a positive example, otherwise it is a negative example;

$$\beta_i = \begin{cases} 1 & \text{if}\, y \in W \\ -1 & \text{if}\, y \notin W \end{cases} .$$

The total weight score of a test instance can then be quantified as

$$\gamma = \frac{\sum\limits_{i=0}^{k} f(\alpha_i) \cdot \beta_i}{\sum\limits_{i=0}^{k} f(\alpha_i)}, \quad \gamma \in [-1, 1] \tag{4}$$

where $f(\alpha_i)$ is a given weight function. The two weight functions tested were the linear map, $f : x \mapsto x$, and the exponential map, $f : x \mapsto \exp(x)$.

4.2 Implementation

4.3 Data sets

Considering the scope of this project, only three data sets were evaluated to see how well the algorithm performed under various challenges. All three of the data sets were collected from the University of California Irvine, Center for Machine Learning and Intelligent Systems' Machine Learning Repository[1]. The data sets consisted of categorical and ordinal features, but to satisfy earlier constraints every feature was treated as categorical and dealt with via one hot encoding. Future work should be done on expanding the algorithm to handle both ordinal and continuous features, that will be touched on in the discussions section.

4.3.1 Tic-tac-toe

The Tic-tac-toe data set[2] has 958 instances with nine features corresponding to the cells of a game of tic-tac-toe. These instances are created by all possible end game where 'x' went first[3]. An instance is classified as 'positive' if 'x' won and 'negative' if 'o' won. An example of the classification of four boards can be seen in figure 1. This data set was chosen because games like tic-tac-toe are defined off of a rigorous rule set. The data set is also purely categorical, the assumption that our algorithm works under.

1.

x	x	x
x	o	o
x	o	o

(pos)

2.

b	b	x
o	x	o
x	b	b

(pos)

3.

o	b	b
o	x	b
o	x	x

(neg)

4.

b	b	o
o	b	o
o	x	x

(neg)

Figure 1: An example instance from the Tic-tac-toe data set, reformatted to be better understood. 'x' and 'o' represent each player and 'b' represent blank spaces.

4.3.2 Car Evaluation

The Car Evaluation data set[4] evaluates 1728 data instances of cars into four classifications of quality: unacceptable, acceptable, good, and very good. The evaluation metrics for the cars are buying price

[1] https://archive.ics.uci.edu/ml/index.php

[2] https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame

[3] inspection of the data sets show some games that are not possible, but do are consistent with who won the game.

[4] https://archive.ics.uci.edu/ml/datasets/car+evaluation

(low, medium, high, very high), maintenance price (low, medium, high, very high), number of doors (2, 3, 4, 5 or more), person capacity (2, 4, more), luggage boot size (small, medium, big), and estimated safety (low, medium, and high). According to the source, there is a known concept structure that can be extracted from the dataset, however these are likely harder to deduce than those of the Tic-tac-toe data set. This data set has all ordinal features, so it is expected that the algorithm will have decreased performance. The distribution of classes is very unbalanced, with unacceptable having 70%, acceptable having 22%, good 4% and very good 4%, so this will also allow testing on if lower class presence influences frequency in final rules.

4.3.3 Breast Cancer Wisconsin

The Breast Cancer Wisconsin data set[5] was tested. Because of points that will be discussed later, the testing was less rigorous on this data set. 699 instances of cancer cells were classified as either benign or malignant with nine features. Each feature was ordinal and spanned from $1 - 10$. It is expected that casting so many values into a categorical space as well as the a complex structure with few data instances will yield bad performance.

4.4 Qualitative comparison of rules

Since the motivation of this algorithm is to represent data instead of provide an alternative model, a qualitative study is motivated. A large range of rule sets were generated and a random sampling of rule sets were taken to qualitatively parse performance. Since the scope of this project is very limited, rigorous testing procedures are not in order. All the data can be seen with the written code on GitHub[6].

4.4.1 Tic-tac-toe

The Tic-tac-toe data was tested with rule set size of $k \in \{0, 1, 10, 11, 22\}$. These values were motivated by 0: only relying on class coverage, 1: half the average number of feature values, rounded down, 10: the number of features plus one for the class, 11: the number of features plus the number of classes, and 22: double the previous number to represent some large number of rules. The required number of instances of a class showing up was $k^* \in \{0, 1\}$. Initially the values were set to $\{0, 1, 2, 3\}$, but a bug in the code meant that only the first two values were actually tested. Finally, to see how the accuracy of the forest played into the performance, the data was tested on forest sizes $\in \{20, 50, 100, 200, 500, 1000, 2000, 5000\}$.

The qualitative characteristics of the rule sets can be summarized in the following points:

1. class coverage becomes more uniform as forests become more accurate,
2. with few classes, there was no difference in requiring class coverage when more rules than the number of classes were used,
3. rules sets on the order of 10 rules were the most understandable,
4. in general, the rules are abstract and difficult to understand without domain knowledge.

To justify these claims, see the two small rule set examples (figures 2a and 2b) and the two larger rule set examples (figures 2c and 2d).

These visualizations were created by interpreting rules in the form of

$$\text{“}(\text{tl} = x) \land \neg(\text{tl} = o) \land \neg(\text{tm} = o) \land (\text{tr} = b) \land (\text{ml} = b) \land \neg(\text{ml} = o) \land (\text{bl} = b) \land \neg(\text{bl} = x) \land (\text{bm} = b) \land \neg(\text{br} = o) \longrightarrow \text{negative .”}$$

[5] https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28original%29

[6] https://github.com/WillJardee/CSCI547/; please excuse the mess, as it is still a work in progress and being commented.

o/x	o/x	o/x
x	o/x	o/b
x/b		

1. (pos)

x/b	o/x	o/x
x/b	o/x	o/x
x/b		

2. (neg)

(a) A Tic-tac-toe rule set from $k = 0$, $k^* = 1$, forest size = 20.

o/x	o/b	
o/x	o/x	o/x
o/x	b	x/b

1. (pos)

o/x		x
b	b	
b	b	o

2. (neg)

(b) A Tic-tac-toe rule set from $k = 0$, $k^* = 1$, forest size = 5000.

x	x	o
b	x	o/x
o/x		o

1. (neg)

x/b	o/x	
x/b	x	x
o		x/b

2. (neg)

o/x	x	o/x
x	x	

3. (neg)

x/b	o/b	o/b
x/b		o/x
o		

4. (neg)

x/b	x	o
b	b	o
x		o

5. (neg)

	o/x	o/x
x	o/b	o/x
o/x	x	o/b

6. (neg)

o/b	o/x	
o/b		b
b	o	o/x

7. (neg)

	b	o
b	x	b
b		o/b

8. (pos)

		x/b
o/x	x/b	o/x
x/b	x	o

9. (pos)

	x/b	o/x
o/x	x/b	o/x
o/x		o

10. (neg)

(c) A Tic-tac-toe rule set from $k = 10$, $k^* = 1$, forest size = 20.

x	x/b	b
b		
o/b	b	o/x

1. (neg)

	o/x	o/x
x		
x	x	

2. (pos)

o/x	b	x
b	x	o
b		b

3. (pos)

x/b	o/x	x/b
o/x	x	o/b
o/x		x

4. (neg)

	b	o
o	b	x
b	x	b

5. (neg)

x/b	o	x
		x
b	b	o/x

6. (pos)

		o/x
o/x	b	o
x/b	x/b	b

7. (pos)

o/x		x/b
b	o	o
b		b

8. (neg)

o/x		x/b
x	o	o
o/x		

9. (neg)

o/x	x/b	o/x
x	o	o
o/x	o	o/x

10. (neg)

(d) A Tic-tac-toe rule set from $k = 10$, $k^* = 1$, forest size = 5000.

Figure 2: A collection of rule sets derived from the Tic-tac-toe data set.

After studying the resulting boards that were returned, one can convince themselves that there is some underlying structure. However, the visualization on boards of this sort doesn't aid in learning an abstract rule like "if three 'x' in a row ↔ positive." There does seem to be some promise in this area with boards like like 2 in figure 2b and number of possible 'x' or 'o' rows in the positive and negative examples, relatively.

4.4.2 Car Evaluation

The tests for Car Evaluation were very similar in construction and reasoning as the Tic-tac-toe data set, with $k \in \{0, 2, 6, 10, 20\}$, $k^* \in \{0, 1\}$, and forest size $\in \{20, 50, 100, 200, 500, 1000, 2000, 5000\}$. The bug related to k^* was still present in these tests.

The Car Evaluation data set rules are much harder to interpret and analyze, returning to point 4 brought up in with the last data set. It seems that points 2, 3, and 4 still hold, but 1 does not. It seemed that in most rule sets analyzed that rules with larger eigenvalues tended to include either unacceptable or very good, the two extremes of the spectrum. This makes intuitive sense, as deliminating extremes in a data set is a much easier task then those near the midpoints. The fact that 70% of the data set consisted of unacceptable and 4% of very good seemed to have a small impact on the rule sets generated from small forests and no notable impact on large forests.

For an example, one rule looked like

$$\text{“}\neg(\text{buying=low})\wedge\neg(\text{maint=vhigh})\wedge(\text{doors=5 more})\wedge\neg(\text{doors=2})\wedge(\text{persons=2})\wedge(\text{lug\_boot=med}) \\ \wedge\neg(\text{lug\_boot=big})\wedge(\text{safety=low})\longrightarrow\text{unacc or vgood .”}$$

It may be easier to interpret if the rules are rewritten as

$$[(\text{doors=5 more})\wedge(\text{persons=2})\wedge(\text{lug\_boot=med})\wedge(\text{safety=low})] \\ \wedge\neg[(\text{buying=low})\vee(\text{maint=vhigh})\vee(\text{doors=2})\vee(\text{lug\_boot=big})]\longrightarrow\text{ unacc or vgood ,}$$

where the required positive features are grouped and the required negative features are grouped. It should be understandable how gathering five to ten of these rules together soon becomes difficult to track. By searching through many sets of rules a patterns do seem to emerge, especially related to unacceptable cars and very good cars. However, this requires many rule sets with different forests and domain knowledge. An example of one of these patterns is the relationship between cars being expensive, having high maintenance costs and low safety scores with either unacceptable cars or very good cars. This can be thought of as the rules attempting to find the patterns present in both older vehicles, which may be unacceptable, and high end sports cars, which are considered very good.

4.4.3 Breast Cancer Wisconsin

The Breast Cancer Wisconsin data set was run with a $k = 10$, $k^* = 1$ and forest size of 20 and 1000, five times. In 4 our of the 5 trials and 3 out of 5 trials for the 20 and 1000 tree forests, respectively, the first 10 rules implied the empty set of classes and the eleventh rule was just “ $\emptyset \longrightarrow$ benign or malignant.” This resulted from the class section of the co-variance matrix being approximately independent from the feature section. The clear interpretation of this is that the underlying rules of the data set are too complex to be picked up by our algorithm as well as the size of the feature space, being on the order of 2^{90} when all the features were considered categorical, being too large for the number of classes. Possible approaches to solving this problem will be touched on in the discussions portion.

4.5 Quantitative study

To test the performance of the data quantitatively, each trial as outlined in the qualitative portion, was split into a 70% trial set and 30% test set. For each instance in the testing set the accuracy measure proposed in equation 4 was used, with a linear weight function and an exponential weight function. The performance among every axis of analysis, k, k^*, and forest size, were practically stochastic, see figure 3 for an example. It is unclear whether this response came from the accuracy metric proposed being an incorrect derivation or generally poor performance of the algorithm. It has already been partially discussed in the qualitative section that patterns didn't seem to become evident until rules over multiple forests were considered. If this were the fact, then it would make sense that the performance against one rule set would be poor. Since the quantitative results don't give any insight into performance, good or bad, the rest of them will be omitted.

5. Discussion

5.1 Contributions and future work

As can be seen from the experimental results, the general pattern of the algorithm was not optimistic. It seems that the algorithm developed was able to pull out some general patterns, especially when patterns were searched for from multiple runs. There was an evidential difference in the rules deduced

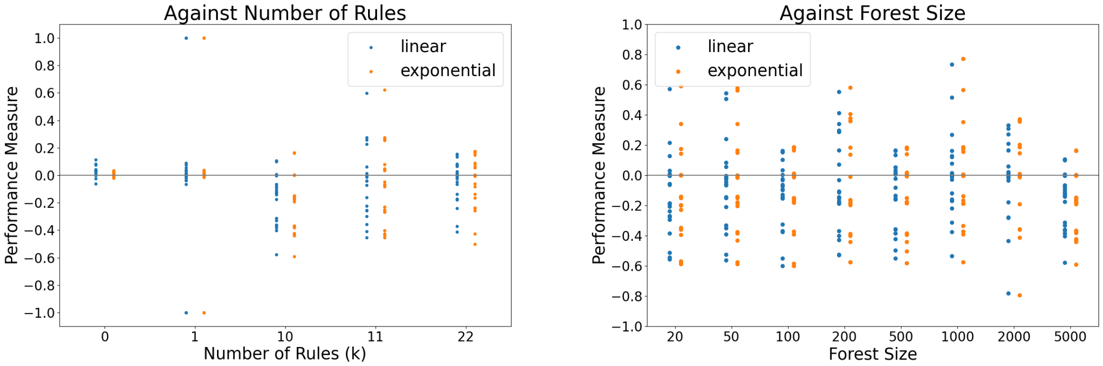

Figure 3: The performance metric, according to equation 4, for the Tic-tac-toe data set. The number of rules and forest size are shown here using a linear weight function (blue/left) and the exponential weight function (orange/right). It can be seen that the performance is stochastic around a small negative value.

from better forests, meaning there is a relationship between the quality of the forest structure and the quality of the rules deduced. It is likely that the algorithm would be improved if a couple key facets were addressed:

1. better statistical analysis of how to construct the co-variance matrix and how to weight rules, rule elements, and classes according to eigen decomposition characteristics,

2. expansion of the algorithm to deal with intervals from ordinal and real valued features,

3. more sophisticated analysis of the quantitative measure function in equation 4, and

4. expansion of the algorithm to consider relationships between rules from the same tree before adding them to the co-variance matrix.

Since this algorithm did not successfully tackle any of these points, it serves as the starting point for a larger project that could build off the concept outlined here.

5.2 Rules as interpretation aid

A lot of issues stemmed from not properly understanding how to use logical rules as an interpretation aid. As discussed in the related works, there are preexisting methods that use rules to understand structures like neural networks. In development, most of the focus was on forest specific literature and other methods that address representing either trees or forests. To improve this method guidance should be taken from those related works to better address how to present the rule set. For example, in theory it seemed logical to present the counter-factuals of ordinal data points, but in testing it added little understanding to the logical rule. If the rules were better displayed and a more rigorously backed measure function used then the fundamental algorithm may have shown better performance.

6. Conclusion

References

Gilbert Strang. *Linear algebra and its applications.* Acad. Press, 4th edition, 2006.