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

2.4 Eigenspaces

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 the path of "$\neg A, \neg B, C$" and returns the class W_0. This decision can be stated as

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

Using material implication, sometimes also referred to as modus ponus, and De Morgan's law this rule can be written as

$$\begin{gathered} \neg\left(\neg A \wedge \neg B \wedge C\right) \vee W_0 \,, \\ \neg(\neg A) \vee \neg(\neg B) \vee \neg(C) \vee W_0 \,. \end{gathered} \tag{1}$$

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, they probably come from the same logical rule. The same can be said about a clause and classification. The pattern here can be described by a co-variance matrix.

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. For simplicity, all of these features are assumed to be binary so that for each feature there are only two values, providing the $2n$ instead of just n. 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 $[\neg(A), \neg(B), \neg(C), A, B, C]$. The last c would correspond to the possible classes. Each time a pair of clauses show up together in a rule in the extracted rule 1 will be added to the corresponding cell.

To illustrate this, see the co-variance matrix, Δ, for the continuing example:

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

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

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. Consequently, the k most important vectors can be chosen by picking the eigenvectors that correspond to the k largest eigenvalues. There is likely small components from many clauses and large components from a few. 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.

Because the distribution of the component values is not known, sophisticated models shouldn't be used here; i.e., a standard deviation cut cannot be done because there is no guarantee the components are Gaussian. For this paper we assigned a 1st quartile cut, where only values in the top 25% are kept. 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 appropriate value is kept, the rest are 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 probablity 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 outcome is, or used in the performance 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 rules 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 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 first quartile of the features will be kept; 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 classes are $(0.5, 0, 0.5, 0)$. 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, for possible future use.

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 theoredical 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

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

Then, the time complexity of creating the rule set will be $\mathcal{O}(l!D)$, the number of paths in a full tree will be on the order of $l!$, even for non-binary trees. This is over all the trees in the forest. 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 $\mathcal{O}(n^{2.3})$. If the number of trees is assumed to be much larger than the number of features, which should be true for an accurate forest, then the complexity of the whole algorithm becomes $\mathcal{O}(n^2 l!D)$.

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 × n matrix of zeros
for rule in extractedRules do
    if feature i and feature j in rule then
        $Map_{ij}$ ← $Map_{ij} + 1$
        $Map_{ji}$ ← $Map_{ji} + 1$
    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
```

4. Experimentation

4.1 Proposed performance metric

To measure the performance of a test set with the derived rule set, a new metric was developed. The motivation of this metric is that for a given input vector, $\mathbf{X}$, related class, y, the importance of rules that the $\mathbf{X}$ fit need to be weighted heavier than those it does not fit. The measurement of how well $\mathbf{X}$ matches up is done through a dot product in Cartesian space (however, there is no reason that a complicated metric/kernel could not be applied to the inner-product). To test the negative nodes, i.e. $\bar{A}$, $\mathbf{X}$ can be subtracted from the vector of all ones, 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 $\bar{A}$, where the negative clauses have a value of 1, 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. This value is easier to see when viewed in the context of one hot encoding, as the potential the product satisfied stochastically will be on the order of $n-1$. As this is specifically in the context of a dot product in Cartesian space, if the metric/kernel is being changed than this value should be too.

To ensure that these two products are on the same magnitude, they 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

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

So, 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 quanitifed as

$$\gamma = \frac{\sum_{i=0}^{k} f(\alpha_i) \cdot \beta_i}{\sum_{i=0}^{k} f(\alpha_i)} \tag{4}$$

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, $f : x \mapsto \exp(x)$. Because of the normalization, $\gamma \in [-1, 1]$.

4.2 Implementation

4.3 Qualitative comparison of rules

4.4 Quantitative results

5. Discussion

6. Conclusion

References