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 Rule sets from decision trees

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

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

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

$$\neg\left(\neg A \wedge \neg B \wedge C\right) \vee W_0 \,,$$

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

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.

3.2 Co-variance matrix from rule sets

3.3 Rule basis of co-variance matrix

3.4 Rule pruning and class-space coverage

3.5 Time complexity

4. Experimentation

4.1 Implementation

4.2 Proposed performance metric

4.3 Qualitative comparison of rules

4.4 Quantitative results

5. Discussion

6. Conclusion

References

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
        $Map_{ij} \leftarrow Map_{ij} + 1$
        $Map_{ji} \leftarrow Map_{ji} + 1$
    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
```