CHAPTER 10: SYNTHESIS

INTREES/PIPELINE ENGINEERING

I. 0verview

Chapter 9 is about "**synthesis**". It recognizes the practical dimension of solving real-world problems, such as building pipelines that combine and streamline a variety of models and operations. This is nonetheless a diverse topic and anticipates many possible alternatives, depending on the particular problems and contexts. To give one example, here, we introduce the method implemented in the R package, InTrees[1], which combines decision tree, random forest, and feature selection such as LASSO. Note that, as an overarching framework, its combinatory power is not limited with these models.

[1] *Deng, H. Interpreting tree ensembles with inTrees. Manuscript available at: https://arxiv.org/abs/1408.5456*

II. InTrees

II.1 Rationale and Formulation

As we have seen that, the linear regression models are advantageous in providing a linear characterization of a multivariate system. Although its ostensible interpretability comes with a question mark, it is undoubtable a more interpretable model than the tree models. Tree models are more like approximating the data and build on their power in capturing complex interactions between variables. Given their different advantages, it is natural to wonder if we could combine both models and get the best of both. From a pragmatic perspective, as Prof. George Box has pointed out, "*all models are wrong, some are useful*", neither model is the truth, but we can always make the model better in terms of some quantitative criterion such as prediction accuracy on testing data or qualitative criterion such as interpretability.

To combine the two models, first, we may notice that the two models employ different semantics: the regression model uses an equation-based semantics, that is more mathematically and formative; the tree model uses a rule-based semantics, which is more intuitive and heuristic. Can we have a hybrid model that combines both semantics? Sure, we could. One approach we can use is to plug in one semantics into another.

For example, while regression model puts variables into the equation, the variables are defined by users. So, what are the variables?

This is the starting point of InTrees. It takes rules as the variables that can be put into a regression model. To get the rules, InTrees harvests the power of random forest to convert the original raw data into a new dataset whose variables are the rules. As we know, rules represent complex interactions between variables. In this way, we have the best parts of both methods, while the rules capture the variable-level patterns in the data, and the regression equation captures the global effects of these patterns in predicting the outcome variable.

II.2 Theory and Method

Consider the following dataset that has 2 predictors and 7 instances as shown in Table 10.1.

Table 10.1: An exemplary dataset with 7 instances

ID	X_1	X_2	Class
1	1	1	C0
2	1	0	C1
3	0	1	C1
4	0	0	C1
5	0	0	C0
6	0	0	C0
7	0	0	C0

Importance scores from the random forest model can provide insights regarding which variables are important, however, it is still not straightforward to understand. For example, the importance scores from random forests applied to the dataset in Table 10.1 are shown in Figure 10.1. We can only know that X_1 and X_2 have similar importance scores, but it is unclear how exactly these variables work together to predict the class.

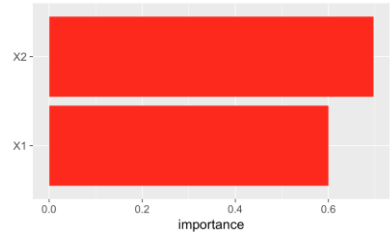

Figure 10.1: Importance score of two variables

To this end, the inTrees framework, illustrated in Figure 10.2, was proposed to extract, measure, prune, and select rules from a tree ensemble. It will further calculate frequent variable interactions, and summarize rules

into a prediction model (rules become the variables). The framework has been implemented in the ``inTrees" R package. In the following we introduce each functionality of the inTrees framework.

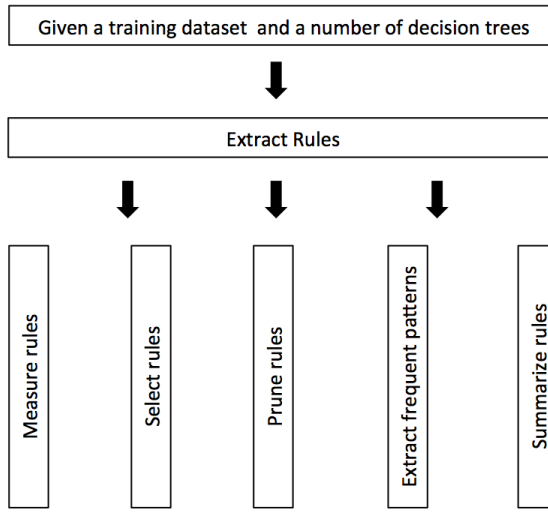

Figure 10.2: The pipeline of inTrees

Rule extraction and measuring: A decision tree can be dissembled into a set of rules. For example, considering the decision tree shown in Figure 10.3, which was created as one tree of the random forest model applied on the dataset shown in Table 1. It can be seen that the tree was built based on the resampled dataset that includes the instances {1,1,2,2,7,7,7}. In other words, in this resampled dataset, the instance (ID: 1) was resampled twice, the instance (ID: 2) was resampled twice, and the instance (ID: 7) was resampled three times. In the tree, the root and inner nodes are labeled with the data point IDs and leaf nodes are labeled with the data point IDs and the decisions (i.e., which class to predict). Here, three rules can be extracted: $\{X_1 = 0 \rightarrow Class = C_1\}$, $\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$, $\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$.

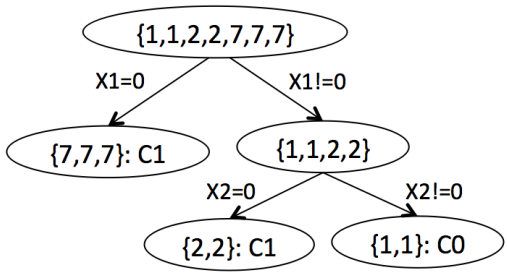

Figure 10.3: An exemplary decision tree

Generally, in a decision tree, a rule can be extracted from the root node to each leaf node as $\{X_{i_1} = a_{i_1}, \ldots, X_{i_k} = a_{i_k} \rightarrow T = t\}$, where X_{i_j} represents the variable used in the path from the root node to a leaf node, $X_{i_j} = a_{i_j}$ represents the criterion for splitting at node i_j, and $T = t$ is the outcome at the leaf node. Note a_{i_j} can be a range when X_{i_j} is numerical and a set of values when it is categorical.

Thus, for a random forest model, we can extract a set of rules by dissembling all its trees. There is one complication, though, that in random forest, as we have mentioned in Chapter 4, each tree is built on a subset of samples and a subset of features in order to be a weak model. That means, we need to revise the outcome of each rule that maybe different from the original outcome associated with this rule provided by the tree. In the inTrees framework, the outcomes from the original rules are ignored. Instead, the outcomes are re-calculated using all the training data, such that the most frequent class is used as the outcome.

For example, for the rule $\{X_1 = 0 \rightarrow Class = C_1\}$ derived from the decision tree shown in Figure 10.3, the three data points (the same instance) have the class of C_1. However, when using all the raining data, there are five data points (IDs: 3-7) satisfying $X_1 = 0$, and 3/5 of the data points have

class of C_0. Therefore, the rule should be updated with the outcome as C_0. The other two rules remain the same since the classes from the tree are consistent with the most frequent classes when applied to all the training data.

After we collect all the rules from the random forest model, the rules are evaluated with three criteria. The **length** of a rule is defined as the number of variable-value pairs in the rule condition. The **frequency** of a rule is the proportion of data points satisfying the rule condition, the left-hand part of the rule. The **error** is the error rate of the rule. For classification problems, it is the number of data points incorrectly identified by the rule divided by the number of data points satisfying the condition. For regression problems, it is mean squared error defined as:

$$err = \frac{1}{k}\sum_{i=1}^{k}(t_i - \tilde{t})^2,$$

where k is the number of data points in the leaf node, t_i is the value of the response variable of the i^{th} data point, and $\tilde{t}$ is the prediction at the leaf node.

For the illustrative example and decision tree, the evaluation of the three rules are shown in the Table 10.2.

Table 10.2: Evaluation of the three rules extracted from the tree shown in Figure 10.3

ID	Rule	length	frequency	error
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	1	5/7	2/5
2	$\{X_1 != 0, X_2 = 0 \rightarrow Class = C_1\}$	2	1/7	0
3	$\{X_1 != 0, X_2 != 0 \rightarrow Class = C_0\}$	1	1/7	0

Prune rules: As each tree in a tree ensemble can be weak, the rules we collect can include irrelevant and redundant variable-value pairs. Therefore, it may be beneficial to remove/prune irrelevant and redundant rules.

Let p_i denote the i^{th} variable-value pair of a rule condition, such that a rule can be written as $< p_1, \ldots, p_k > \rightarrow T = t$. To prune the rule, inTrees uses leave-one-out pruning, that is, at each round, removes one pair and checks how much error this removal will induce. The pair with the least error increase is removed, if it is also below a pre-specified threshold. The increase of error is referred as the **decay** in the terminology of inTrees.

Two types of decay are defined. The first one is the absolute error increase, defined as:

$$decay_i = Err_{-i} - Err.$$

The second one is the relative error increase, defined as:

$$decay_i = \frac{Err_{-i} - Err}{\max(Err, s)},$$

where Err is the error of the original rule, Err_{-i} is the error of the rule with the p_i removed, and s is a small positive constant (e.g., 0.001) that bounds the value of $decay$ when Err is zero or close to zero.

Take rule $\{X_1 != 0, X_2 = 0 \rightarrow Class = C_1\}$ for example. The error for this rule is known to be 0 if we check the illustrative dataset aforementioned. Assume that the threshold is 0.05. Now remove $X_1 != 0$ from the rule condition, and the new rule becomes $\{X_2 = 0 \rightarrow Class = C_1\}$, which has an error of 3/5. Therefore, the absolute error increase is 3/5. Therefore, $X_1 != 0$ should not be pruned.

On the other hand, we can also see that the relative error increase is $\frac{3}{5*s}$. With a default value as $s = 0.001$, the relative error increase is also large, indicating that $X_1 != 0$ should not be pruned.

Now let's remove $X_2 = 0$. The resulting rule $\{X_1 != 0 \rightarrow Class = C_1\}$ has an error of 1/2. Therefore, removing $X_2 = 0$ also hurts the accuracy of the rule. The rule should not be pruned.

Let's do another example. Suppose that the rules are built with the following data set shown in Table 10.3.

Table 10.3: An exemplary dataset

ID	X_1	X_2	Class
1	1	0	C_1
2	1	0	C_1
3	1	1	C_0
4	0	1	C_0

On this dataset, the rule $\{X_1 != 0, X_2 = 0 \rightarrow Class = C_1\}$ has an error of 0. The error after removing $X_1 != 0$ is still 0, therefore, both the absolute and relative error increase are 0. The error after removing $X_2 = 0$ becomes 1/4. Therefore, the absolute error increase is 0.25, and the relative error increase is $\frac{1}{4*s}$. Thus, $X_1 = 0$ should be removed, and the new pruned rule becomes $\{X_2 = 0 \rightarrow Class = C_1\}$. This pruning process will continue. After removing $X_2 = 0$ from the rule, the rule is effectively a random guess, thus the error becomes 0.5. This indicates that no variable-value pair can be removed, the pruning should be stopped, and the final rule is $\{X_2 = 0 \rightarrow Class = C_1\}$.

Select rules: In previous sections, each rule is pruned and the pruned rules can be ranked by accuracy, frequency and complexity. However, a tree ensemble can generate numerous rules, and many of them can be redundant. If the top rules tend to be redundant rules, the ranked rule set is

again hard to interpret. Therefore, selecting a non-redundant rule set is valuable for interpretation.

The inTrees framework casts the rule selection problem into the feature selection formulation. This is built on the creation of a new dataset based on the rule set that binarizes the original dataset. For example, continue our discussion of the dataset with 7 instances as shown in Table 10.1, the three rules we have collected are shown in Table 10.4.

Table 10.4: The three rules

ID	Rule
1	$\{X_1 = 0 \rightarrow Class = C_0\}$
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$

Table 10.5: The binarized dataset of Table 10.1 by the rules in Table 10.4

ID	Z_1	Z_2	Z_3	Class
1	0	0	1	C0
2	0	1	0	C1
3	1	0	0	C1
4	1	0	0	C1
5	1	0	0	C0
6	1	0	0	C0
7	1	0	0	C0

Consider $\{X_1 = 0 \rightarrow Class = C_0\}$ first. In the new dataset, a binary feature Z_1 is created for the condition. The instances satisfying the condition $X_1 = 0$ include {3,4,5,6,7} and therefore, the binary feature values of the instances are {0,0,1,1,1,1,1}. For $\{X_1! = 0, X_2 = 0 \rightarrow$

$Class = C_1\}$, only instance 2 satisfies the condition, and therefore, the binary feature values of the instances for the second rule is {0,1,0,0,0,0,0}. Similarly, the feature values for the third rule is {1,0,0,0,0,0,0}. Following this process, the new converted data set is shown in Table 10.5.

Now a feature selection method can be applied on this new data set. The goal of feature selection is to select a subset of relevant but not redundant features from the original features. Feature selection methods include L_1 regularized logistics regression (or LASSO) and regularized random forests are used in the inTrees R package. Suppose a feature selection method selects the feature subset as $\{Z_1, Z_2\}$. It indicates that only the first and second rules should be used in prediction.

Note that the binary feature representation does not include the information about the length of rules. Given two rules with the same predictive power, the rule with a smaller length may be preferred in terms of interpretability. In the inTrees framework, the guided regularized random forests are used. The guided regularized random forests (GRRF) can assign a weight to each feature, so that when two features have similar predictive power, the feature with more weight is more likely to be selected. In this case, shorter rules are more likely to be selected.

Frequent variable interaction: The inTrees framework further provides extraction of variable interactions that have been important in the selected rules. A rule essentially encodes these interaction information, e.g., the rule $\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$ captures the interaction between X_1 and X_2.

Consider the following set of rules extracted by inTrees. Association rule analysis is used for mining the frequent variable-value pairs from the rules. In particular, each variable-value pair is considered as an item.

First, let's consider all the rule conditions (left-hand side of the rules). Define the **support** of a particular interaction pattern (e.g., X_1 and X_2) as

the number of rules that encode this interaction in the rule set, divided by the size of the rule set (size = 4 for the illustrative example). For example, the support of each variable interaction pattern extracted from the rules in Table 10.6 is calculated in Table 10.7.

Table 10.6: An exemplary set of rules

ID	Rule
1	$\{X_1 = 0 \rightarrow Class = C_0\}$
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$
4	$\{X_1! = 0, X_2 = 0, X_3 = 1 \rightarrow Class = C_1\}$

Table 10.7: Interaction patterns extracted from the rules in Table 5 and their supports

Variable-value pairs (interaction patterns)	Support
$X_1 = 0$	1/4
$X_1! = 0$	3/4
$X_2 = 0$	1/4
$X_3 = 1$	1/4
$X_1! = 0, X_2 = 0$	2/4
$X_1! = 0, X_2! = 0$	1/4
$X_1! = 0, X_2 = 0, X_3 = 1$	1/4

Hence, the most frequent interactions are $X_1! = 0$ and $X_1! = 0, X_2 = 0$. Now consider the right hand of each interaction. The **confidence** is defined as the accuracy of an interaction pattern predicting a particular class. For example, continuing the example mentioned above, the confidence of the interactions can be calculated as shown in Table 10.8.

Table 10.8: Interaction patterns extracted from the rules in Table 10.6 and their confidences

Variable-value pairs (interaction patterns)	Class	Confidence
$X_1 = 0$	$Class = C_0$	1/1
$X_1! = 0$	$Class = C_1$	2/3
$X_2 = 0$	$Class = C_1$	1/1
$X_3 = 1$	$Class = C_1$	1/4
$X_1! = 0, X_2 = 0$	$Class = C_1$	2/2
$X_1! = 0, X_2! = 0$	$Class = C_0$	1/1
$X_1! = 0, X_2 = 0, X_3 = 1$	$Class = C_1$	1/1

Combine both Tables, we can see the top variable interactions in terms of confidence and support is

$$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}.$$

with a support of 0.5 and confidence of 1. This indicates this variable interaction plays an important role in the tree ensemble.

Note that, for continuous features, it may be useful to do a discretization before inputting the data into inTrees. This is because that there can be many possible splitting points for continuous features in a tree, resulting in the possibility that fewer frequent variable interactions for continuous features could be identified.

Summarize rules: Once rules from tree ensembles are pruned and selected, we can summarize these high-quality rules into classifiers. There are multiple methods for summarizing. For example, the method RuleFit[1] used a linear model for summarizing the rules. Here, we introduce a simple method to summarize the rules into an ordered rule set for prediction.

[1] *Friedman, J.H. and Popescu, B.E. Predictive learning via rule ensembles. Annals of applied statistics, 2008.*

The method has multiple iterations. Denote r_0 as the default rule (i.e., with null condition) that classifies all data points to the most frequent class. Denote the ordered rule set as R, which is set to be empty at the beginning. Then, at each iteration, the best rule in the rule set is selected and added to R. The best rule is defined as the rule with the minimum error evaluated by the training data. If there are ties, the rule with higher frequency and smaller length is selected. Then, the data points that satisfy the condition of the best rule are removed, and the default rule r_0 is re-calculated with the data points left. This iterative process continues until no instance is left in the training dataset, or the default rule r_0 has the best accuracy comparing with other rules in the remaining rule set. Note that, the selected rules in R are ordered according to the sequence of their inclusion.

Consider the dataset shown in Table 1 and the rule set shown in Table 4. At the beginning, the default rule is $\{Class = C_0\}$ with error rate of 3/7. The error rate and frequency of each rule is shown in Table 10.9.

Table 10.9: Error rates and frequencies of the rules in Table 10.5 using dataset in Table 10.1

ID	Rule	Error	Frequency
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	2/5	5/7
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$	0/1	1/7
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$	0/1	1/7
4	$Class = C_0$	3/7	

In this case, Rule 1 and Rule 2 have the least errors, and their frequency and length are also the same. Thus, we can select either of them to the ordered rule set $R = \{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$. Then, the data point (ID:2) classified by this rule is removed. The default rule is still

$\{Class = C_0\}$, and the error and frequency of each rule on the new dataset is updated in Table 10.10.

Table 10.10: Updated error rates and frequencies of the rules in Table 10.5 using the reduced dataset in Table 10.1 (data point ID:2 is removed)

ID	Rule	Error	Frequency
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	2/5	5/6
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$	NA	0/6
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$	0/1	1/6
4	$Class = C_0$	2/6	

Table 10.11: Updated error rates and frequencies of the rules in Table 4 using the reduced dataset in Table 1 (data points ID:1 and ID:2 are removed)

ID	Rule	Error	Frequency
1	$\{X_1 = 0 \rightarrow Class = C_0\}$	2/5	5/5
2	$\{X_1! = 0, X_2 = 0 \rightarrow Class = C_1\}$	NA	0/5
3	$\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$	NA	0/5
4	$Class = C_0$	2/5	

Then, $\{X_1! = 0, X_2! = 0 \rightarrow Class = C_0\}$ is added to R, and the data point (ID:1) is removed. The default rule remains unchanged and the error and frequency of each rule on the new dataset is updated in Table 10.11.

Now the default rule $Class = C_0$ has the minimum error 2/5, the same as $\{X_1 = 0 \rightarrow Class = C_0\}$. Therefore, the default rule is added to R and the process stops. The final ordered rule set R is summarized in Table 10.12.

Table 10.12: Final results of R

Order	Rule
1	$\{X_1 != 0, X_2 = 0 \rightarrow Class = C_1\}$
2	$\{X_1 != 0, X_2 != 0 \rightarrow Class = C_0\}$
3	$Class = C_0$

When predicting on an instance, the first rule in R satisfying the data point is used for prediction. For example, for a data point $\{X_1 = 0, X_2 = 1\}$, it does not satisfy neither Rule 1 or Rule 2 in R. Therefore, the default rule is used, and the prediction is C_0.

II.3 R Lab

Here we apply random forests to the AD dataset and use `inTrees` to extract rules. First, based on the random forest model, 4555 rules are extracted.

```
rm(list = ls(all = TRUE))
library("arules")
library("randomForest")
library("RRF")
library("inTrees")
library("reshape")
library("ggplot2")
set.seed(1)
path <- "../../data/AD_bl.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_bl")
target <- paste0("class_", as.character(data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("DX_bl", "ID", "TOTAL13",
"MMSCORE"))
X <- data
X <- X[, -rm_indx]
for (i in 1:ncol(X)) X[, i] <- as.factor(dicretizeVector(X[, i],
K = 3))

rf <- randomForest(X, as.factor(target))

treeList <- RF2List(rf)  # transform rf object to an inTrees' for
```

```
mat
exec <- extractRules(treeList, X)  # R-executable conditions

## 4555 rules (length<=6) were extracted from the first 100 trees.
```

Next, the rules are measured by length, error and frequency. E.g., the statistics of 5 rules are shown in the graph below.

```
class <- paste0("class_", as.character(target))
rules <- getRuleMetric(exec, X, target)
print(rules[order(as.numeric(rules[, "len"])), ][1:5, ])

##      len freq    err     condition
## [1,] "2" "0.118" "0.098" "X[,6] %in% c('L1') & X[,11] %in% c('L1')"
## [2,] "2" "0.182" "0"     "X[,4] %in% c('L1') & X[,6] %in% c('L1')"
## [3,] "2" "0.182" "0"     "X[,4] %in% c('L1') & X[,6] %in% c('L1')"
## [4,] "2" "0.081" "0.024" "X[,3] %in% c('L3') & X[,4] %in% c('L3')"
## [5,] "2" "0.043" "0.136" "X[,6] %in% c('L3') & X[,7] %in% c('L3')"
##      pred
## [1,] "class_1"
## [2,] "class_1"
## [3,] "class_1"
## [4,] "class_0"
## [5,] "class_0"
```

For rule pruning, first, we try with absolute decay using a threshold of 0.005 (`maxDecay = 0.005`) to prune the rules, that is, a variable-pair is not removed if the decay is larger than 0.005. The statistics of the rules before and after pruning are shown in Figures 4-6.

The R code below generates Figure 10.4.

```
rules.pruned <- pruneRule(rules, X, target, maxDecay = 0.005, typ
eDecay = 2)

length <- data.frame(original = as.numeric(rules[, "len"]), prune
d = as.numeric(rules.pruned[,
    "len"]))
ggplot(melt(length), aes(value, fill = variable)) + geom_histogra
m(position = "dodge",
    binwidth = 0.4) + ggtitle("Histogram of Lengths") + theme(plo
t.title = element_text(hjust = 0.5))
```

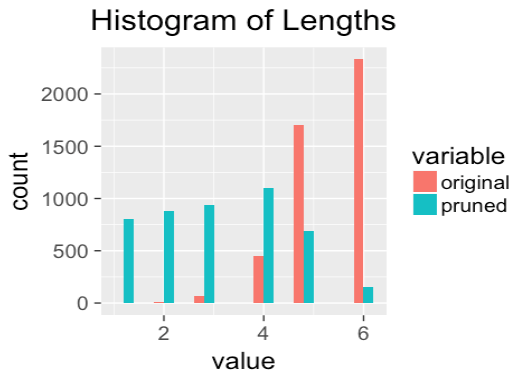

Figure 10.4: Histogram of lengths of the rules before and after the pruning

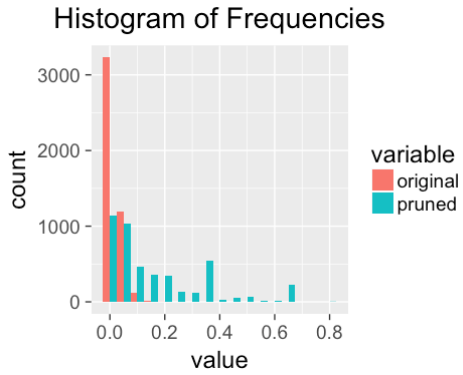

Figure 10.5: Histogram of frequencies of the rules before and after the pruning

The R code below generates Figure 10.5.

```
frequency <- data.frame(original = as.numeric(rules[, "freq"]), p
runed = as.numeric(rules.pruned[,
    "freq"]))
ggplot(melt(frequency), aes(value, fill = variable)) + geom_histo
gram(position = "dodge",
    binwidth = 0.05) + ggtitle("Histogram of Frequencies") + them
e(plot.title = element_text(hjust = 0.5))
```

The R code below generates Figure 10.6.

```
error <- data.frame(original = as.numeric(rules[, "err"]), pruned
 = as.numeric(rules.pruned[,
    "err"]))
ggplot(melt(error), aes(value, fill = variable)) + geom_histogram
(position = "dodge",
    binwidth = 0.01) + ggtitle("Histogram of Errors") + theme(plo
t.title = element_text(hjust = 0.5))
```

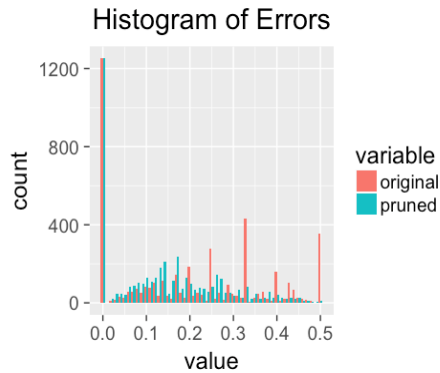

Figure 10.6: Histogram of errors of the rules before and after the pruning

It can be seen that, the lengths of rules are substantially reduced. For example, a majority of the original rules have length of 6 (as the default max length is set to be 6), while after pruning, only a slight percentage of the rules have length of 6. Also, since rules are shortened, the reduction of frequencies are also significant. In terms of errors, after pruning, the error distribution has shifted to the left. Therefore, the rules are simplified without significant sacrifice of accuracy.

For a comparison, we conduct one more experiment using the relative decay with a threshold of 0.05 (`maxDecay = 0.05`). Results are shown in Figures 7-9.

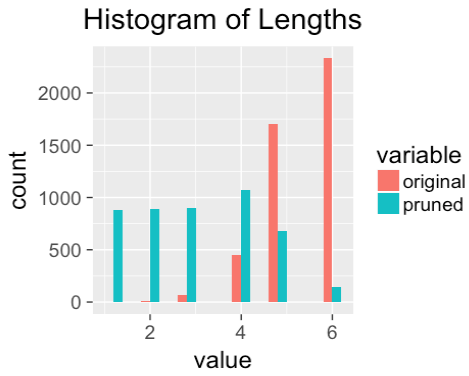

Figure 10.7: Histogram of lengths of the rules before and after the pruning

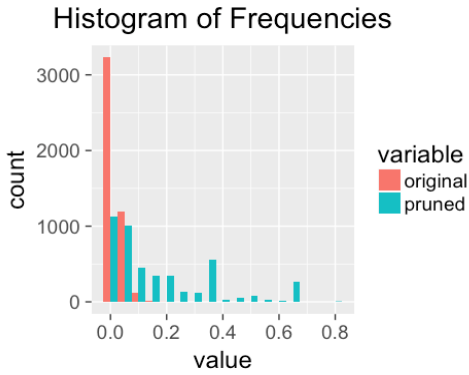

Figure 10.8: Histogram of frequenties of the rules before and after the pruning

The R code below generates Figure 10.7.

```
rules.pruned <- pruneRule(rules, X, target, maxDecay = 0.05, type
Decay = 1)

length <- data.frame(original = as.numeric(rules[, "len"]), prune
d = as.numeric(rules.pruned[,
    "len"]))
ggplot(melt(length), aes(value, fill = variable)) + geom_histogra
m(position = "dodge",
```

```
    binwidth = 0.4) + ggtitle("Histogram of Lengths") + theme(plo
t.title = element_text(hjust = 0.5))
```

The R code below generates Figure 10.8.

```
frequency <- data.frame(original = as.numeric(rules[, "freq"]), p
runed = as.numeric(rules.pruned[,
    "freq"]))
ggplot(melt(frequency), aes(value, fill = variable)) + geom_histo
gram(position = "dodge",
    binwidth = 0.05) + ggtitle("Histogram of Frequencies") + them
e(plot.title = element_text(hjust = 0.5))
```

The R code below generates Figure 10.9.

```
error <- data.frame(original = as.numeric(rules[, "err"]), pruned
 = as.numeric(rules.pruned[,
    "err"]))
ggplot(melt(error), aes(value, fill = variable)) + geom_histogram
(position = "dodge",
    binwidth = 0.01) + ggtitle("Histogram of Errors") + theme(plo
t.title = element_text(hjust = 0.5))
```

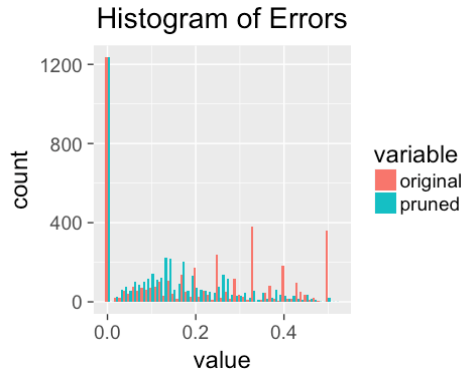

Figure 10.9: Histogram of errors of the rules before and after the pruning

The changes of lengths, frequencies and errors, look similar to the previous results using the absolute decay. An advantage of using relative decay is that one does not need to know the baseline error of a dataset. However, relative decay depends on the baseline error of each original rule,

and when the baseline error is small, e.g., 0, the relative error increase can be large even if the absolute error increase is small.

Now let's consider rule selection. The following R codes applies rule selection to the pruned rule set.

```
rules.selected <- selectRuleRRF(rules.pruned, X, target)
rules.present <- presentRules(rules.selected, colnames(X))
print(cbind(ID = 1:nrow(rules.present), rules.present[, c("condit
ion", "pred")]))
```

After selection, only 16 rules are selected.

```
##       ID
##  [1,] "1"
##  [2,] "2"
##  [3,] "3"
##  [4,] "4"
##  [5,] "5"
##  [6,] "6"
##  [7,] "7"
##  [8,] "8"
##  [9,] "9"
## [10,] "10"
## [11,] "11"
## [12,] "12"
## [13,] "13"
## [14,] "14"
## [15,] "15"
## [16,] "16"
##       condition

#  [1,] "FDG %in% c('L1','L2') & HippoNV %in% c('L1')"

#  [2,] "FDG %in% c('L1') & HippoNV %in% c('L1','L2')"

#  [3,] "PTGENDER %in% c('L2') & FDG %in% c('L2') & AV45 %in% c('
L1','L2') & rs3818361 %in% c('L2') & rs3851179 %in% c('L2')"
#  [4,] "AGE %in% c('L3') & FDG %in% c('L1') & HippoNV %in% c('L1
','L2')"
#  [5,] "PTEDUCAT %in% c('L1') & AV45 %in% c('L2') & HippoNV %in%
 c('L2') & rs610932 %in% c('L2')"
#  [6,] "HippoNV %in% c('L1') & rs3818361 %in% c('L1')"

#  [7,] "AV45 %in% c('L3') & HippoNV %in% c('L1','L2')
#  [8,] "AV45 %in% c('L1','L2') & HippoNV %in% c('L2') && rs37646
```

```
50 %in% c('L1')"
#  [9,] "AGE %in% c('L1') & PTGENDER %in% c('L2') & FDG %in% c('L
2') & AV45 %in% c('L1','L2') & HippoNV %in% c('L1')"
# [10,] "AGE %in% c('L3') & PTGENDER %in% c('L1') & PTEDUCAT %in%
 c('L2') & AV45 %in% c('L1','L2')"
# [11,] "AGE %in% c('L2') & PTEDUCAT %in% c('L2','L3') & HippoNV
 %in% c('L2','L3') & e4_1 %in% c('L2')"
# [12,] "AGE %in% c('L2') & PTEDUCAT %in% c('L3') & e4_1 %in% c('
L1')"
# [13,] "PTEDUCAT %in% c('L1','L3') & e4_1 %in% c('L1') & rs11136
000 %in% c('L1') & rs610932 %in% c('L1')"
# [14,] "AGE %in% c('L2') & HippoNV %in% c('L2','L3') & rs3865444
 %in% c('L1')"
# [15,] "AGE %in% c('L1','L2') & AV45 %in% c('L1')"
# [16,] "PTEDUCAT %in% c('L1','L3') & FDG %in% c('L2','L3') & Hip
poNV %in% c('L2','L3')"


##       pred
##  [1,] "class_1"
##  [2,] "class_1"
##  [3,] "class_0"
##  [4,] "class_1"
##  [5,] "class_1"
##  [6,] "class_1"
##  [7,] "class_1"
##  [8,] "class_0"
##  [9,] "class_0"
## [10,] "class_0"
## [11,] "class_0"
## [12,] "class_0"
## [13,] "class_0"
## [14,] "class_0"
## [15,] "class_0"
## [16,] "class_0"

print(cbind(ID = 1:nrow(rules.present), rules.present[, c("len",
"freq", "err")]))

##       ID   len freq    err
##  [1,] "1"  "2" "0.279" "0.083"
##  [2,] "2"  "2" "0.279" "0.09"
##  [3,] "3"  "5" "0.029" "0.133"
##  [4,] "4"  "3" "0.122" "0.016"
##  [5,] "5"  "4" "0.031" "0.312"
##  [6,] "6"  "2" "0.207" "0.121"
##  [7,] "7"  "3" "0.172" "0.124"
```

```
##  [8,] "8"  "4" "0.06"  "0.194"
##  [9,] "9"  "5" "0.006" "0"
## [10,] "10" "4" "0.044" "0.13"
## [11,] "11" "5" "0.019" "0.2"
## [12,] "12" "3" "0.043" "0.182"
## [13,] "13" "4" "0.037" "0.158"
## [14,] "14" "3" "0.114" "0.203"
## [15,] "15" "2" "0.234" "0.215"
## [16,] "16" "3" "0.282" "0.144"
```

Now let's extract the frequent variable interactions by the function `getFreqPattern()`. Here, we discretize the continuous features to 3 levels with equal frequency.

```
freqPattern <- getFreqPattern(rules.pruned)

top.pattern <- (freqPattern[which(as.numeric(freqPattern[, "len"])
 >= 2), ][1:5, ])
print(presentRules(top.pattern, colnames(X)))
```

And the top frequency variable interactions (with length greater than 2) are shown below.

```
##      len sup     conf
## [1,] "2" "0.038" "1"
## [2,] "2" "0.026" "1"
## [3,] "2" "0.023" "0.991"
## [4,] "2" "0.022" "0.953"
## [5,] "2" "0.021" "0.99"
##      condition

## [1,] "FDG %in% c('L2','L3') & HippoNV %in% c('L2','L3')"
## [2,] "AV45 %in% c('L1','L2') & HippoNV %in% c('L2','L3')"
## [3,] "HippoNV %in% c('L1') & rs3818361 %in% c('L1')"
## [4,] "AV45 %in% c('L3') & HippoNV %in% c('L1')"
## [5,] "rs610932 %in% c('L1') & HippoNV %in% c('L2','L3')"

##       pred
##  [1,] "class_0"
##  [2,] "class_0"
##  [3,] "class_1"
##  [4,] "class_1"
##  [5,] "class_0"
```

An ordered rule set can be built using the selected rules by the function `buildLearner()`.

```
learner <- buildLearner(rules.selected, X, target)
learner.readable <- presentRules(learner, colnames(X))
print(cbind(ID = 1:nrow(learner.readable), learner.readable[, c("
condition", "pred")]))

##       ID
##  [1,] "1"
##  [2,] "2"
##  [3,] "3"
##  [4,] "4"
##  [5,] "5"
##  [6,] "6"
##  [7,] "7"
##  [8,] "8"
##  [9,] "9"
## [10,] "10"
## [11,] "11"
## [12,] "12"
##       condition

#  [1,] "AGE %in% c('L3') & FDG %in% c('L1') & HippoNV %in% c('L1
','L2')"
#  [2,] "FDG %in% c('L1','L2') & HippoNV %in% c('L1')"
#  [3,] "AGE %in% c('L2') & PTEDUCAT %in% c('L3') & e4_1 %in% c('
L1')"
#  [4,] "PTEDUCAT %in% c('L1','L3') & e4_1 %in% c('L1') & rs11136
000 %in% c('L1') & rs610932 %in% c('L1')"
#  [5,] "AGE %in% c('L3') & PTGENDER %in% c('L1') & AV45 %in% c('
L1','L2')"
#  [6,] "PTGENDER %in% c('L2') & FDG %in% c('L2') & AV45 %in% c('
L1','L2') & rs3818361 %in% c('L2') & rs3851179 %in% c('L2')"
#  [7,] "PTEDUCAT %in% c('L1','L3') & FDG %in% c('L2','L3') & Hip
poNV %in% c('L2','L3')"
#  [8,] "AV45 %in% c('L1','L2') & HippoNV %in% c('L2') & & rs3764
650 %in% c('L1')"
#  [9,] "FDG %in% c('L1') & HippoNV %in% c('L1','L2')"
# [10,] "AGE %in% c('L2') & HippoNV %in% c('L2','L3') & rs3865444
 %in% c('L1')"
# [11,] "AGE %in% c('L1','L2') & AV45 %in% c('L1')"
# [12,] "Else"

##       pred
##  [1,] "class_1"
##  [2,] "class_1"
```

```
##  [3,] "class_0"
##  [4,] "class_0"
##  [5,] "class_0"
##  [6,] "class_0"
##  [7,] "class_0"
##  [8,] "class_0"
##  [9,] "class_1"
## [10,] "class_0"
## [11,] "class_0"
## [12,] "class_0"

print(cbind(ID = 1:nrow(learner.readable), learner.readable[, c("
len", "freq", "err")]))

##       ID   len freq                 err
##  [1,] "1"  "3" "0.121856866537718"  "0.0158730158730159"
##  [2,] "2"  "2" "0.195357833655706"  "0.118811881188119"
##  [3,] "3"  "3" "0.034816247582205"  "0.0555555555555556"
##  [4,] "4"  "4" "0.02321083172147"   "0.0833333333333334"
##  [5,] "5"  "4" "0.0367504835589942" "0.105263157894737"
##  [6,] "6"  "5" "0.0154738878143133" "0.125"
##  [7,] "7"  "3" "0.2321083172147"    "0.158333333333333"
##  [8,] "8"  "4" "0.0212765957446809" "0.181818181818182"
##  [9,] "9"  "2" "0.0328820116054159" "0.0588235294117647"
## [10,] "10" "3" "0.0425531914893617" "0.181818181818182"
## [11,] "11" "2" "0.0851063829787234" "0.204545454545455"
## [12,] "12" "1" "0.158607350096712"  "0.317073170731707"
```

IV. Exercises

Data analysis

1. Find 10 regression datasets from the UCI data repository or R datasets. Use inTrees to do analysis. Identify the final list of rules.

Programming

2. Simulate a dataset that has 10 variables, and design some interactions among the 10 variables (the form of the interactions is open-ended, e.g., it could be rule-based interactions, or any other statistical interactions). You can learn more from the simulation

study in this paper[1] to help you conduct this simulation. Implement inTrees to see if the interactions can be captured.

3. Increase the number of variables to be 100. Make the interaction patterns sparse, e.g., only 20 variable interactions. Implement inTrees to see if the interactions can be captured.

[1] *Friedman, J.H. and Popescu, B.E. Predictive learning via rule ensembles. Annals of applied statistics, 2008*.