Exercises for *Applied Predictive Modeling* Chapter 8 — Regression Trees and Rule–Based Models

Max Kuhn, Kjell Johnson

Version 1
January 8, 2015

The solutions in this file uses several R packages not used in the text. To install all of the packages needed for this document, use:

```
> install.packages(c("AppliedPredictiveModeling", "caret", "Cubist", "ipred",
+                    "mlbench", "party", "randomForest"))
```

Exercise 1

Recreate the simulated data from Exercise 7.2:

```
> library(mlbench)
> set.seed(200)
> simulated <- mlbench.friedman1(200, sd = 1)
> simulated <- cbind(simulated$x, simulated$y)
> simulated <- as.data.frame(simulated)
> colnames(simulated)[ncol(simulated)] <- "y"
```

(a) Fit a random forest model to all of the predictors, then estimate the variable importance scores:

```
> library(randomForest)
> library(caret)
> model1 <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)
> rfImp1 <- varImp(model1, scale = FALSE)
```

Did the random forest model significantly use the uninformative predictors (V6 – V10)?

(b) Now add an additional predictor that is highly correlated with one of the informative predictors. For example:

```
> set.seed(600)
> simulated$duplicate1 <- simulated$V1 + rnorm(200) * .1
> cor(simulated$duplicate1, simulated$V1)
```

```
[1] 0.9458
```

Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictors that is also highly correlated with V1?

(c) Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can be used to calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in (?). Do these importances show the same pattern as the traditional random forest model?

(d) Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur?

Solutions

The predictor importance scores for the simulated data set in part (a) can be seen in Table 1. The model places most importance on predictors 1, 2, 4, and 5, and very little importance on 6 through 10.

Next we will add a highly correlated predictor (Part (b)) and re-model the data. Table 2 lists the importance scores for predictors V1-V10 when we inclucde a predictor that is highly correlated with V1. Notice that the importance score drops for V1 when a highly correlated predictor is included in the data. Predictor V1 has dropped to third in overall importance rank.

```
> model2 <- randomForest(y ~ ., data = simulated, importance = TRUE, ntree = 1000)
> rfImp2 <- varImp(model2, scale = FALSE)
>
> vnames <- c('V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'duplicate1')
>
```

	Overall
V1	8.66
V2	6.37
V3	0.77
V4	7.95
V5	2.10
V6	0.19
V7	0.04
V8	-0.08
V9	-0.06
V10	-0.04

Table 1: Variable importance scores for part (a) simulation.

```
> names(rfImp1) <- "Original"
> rfImp1$Variable <- factor(rownames(rfImp1), levels = vnames)
>
> names(rfImp2) <- "Extra"
> rfImp2$Variable <- factor(rownames(rfImp2), levels = vnames)
>
> rfImps <- merge(rfImp1, rfImp2, all = TRUE)
> rownames(rfImps) <- rfImps$Variable
> rfImps$Variable <- NULL
```

	Original	Extra
V1	8.66	6.21
V2	6.37	6.58
V3	0.77	0.53
V4	7.95	7.06
V5	2.10	2.11
V6	0.19	0.14
V7	0.04	-0.01
V8	-0.08	-0.06
V9	-0.06	-0.04
V10	-0.04	0.00
duplicate1		3.97

Table 2: Variable importance scores for part (b) simulation.

Next, we will build a conditional inference random forest for the original data set and compute the corresponding predictor importance scores. We will also build a conditional inference random forest on the data set that includes the highly correlated extra predictor with V1.

```
> library(party)
> set.seed(147)
> cforest1 <- cforest(y ~ ., data = simulated[, 1:11],
+                     controls = cforest_control(ntree = 1000))
> set.seed(147)
> cforest2 <- cforest(y ~ ., data = simulated,
+                     controls = cforest_control(ntree = 1000))
>
> cfImps1 <- varimp(cforest1)
> cfImps2 <- varimp(cforest2)
> cfImps3 <- varimp(cforest1, conditional = TRUE)
> cfImps4 <- varimp(cforest2, conditional = TRUE)
>
> cfImps1 <- data.frame(Original = cfImps1,
+                       Variable = factor(names(cfImps1), levels = vnames))
>
> cfImps2 <- data.frame(Extra = cfImps2,
+                       Variable = factor(names(cfImps2), levels = vnames))
>
> cfImps3 <- data.frame(CondInf = cfImps3,
+                       Variable = factor(names(cfImps3), levels = vnames))
>
> cfImps4 <- data.frame("CondInf Extra" = cfImps4,
+                       Variable = factor(names(cfImps4), levels = vnames))
>
> cfImps <- merge(cfImps1, cfImps2, all = TRUE)
> cfImps <- merge(cfImps, cfImps3, all = TRUE)
> cfImps <- merge(cfImps, cfImps4, all = TRUE)
> rownames(cfImps) <- cfImps$Variable
> cfImps$Variable <- factor(cfImps$Variable, levels = vnames)
> cfImps <- cfImps[order(cfImps$Variable),]
> cfImps$Variable <- NULL
```

Predictor importance scores for the conditional inference random forests can be seen in Table 3. The conditional inference model has a similar pattern of importance as the random forest model from Part (a), placing most importance on predictors 1, 2, 4, and 5 and very little importance on 6 through 10. Adding a highly correlated predictor has a detrimenal effect on the importance for `V1` dropping its importance rank to third.

Finally, we will examine the effect of adding a highly correlated predictor on bagging and Cubist. We will explore bagging through the following simulation:

```
> library(ipred)
> set.seed(147)
> bagFit1 <- bagging(y ~ ., data = simulated[, 1:11], nbag = 50)
> set.seed(147)
> bagFit2 <- bagging(y ~ ., data = simulated, nbag = 50)
> bagImp1 <- varImp(bagFit1)
> names(bagImp1) <- "Original"
```

	Original	Extra	CondInf	CondInf.Extra
V1	9.16	6.26	3.00	1.37
V2	6.75	6.66	4.07	3.89
V3	0.06	0.09	0.02	0.03
V4	8.76	8.50	4.79	5.37
V5	2.13	2.03	0.76	0.77
V6	-0.00	0.01	0.01	0.01
V7	0.09	-0.00	0.03	0.00
V8	-0.06	-0.08	-0.02	-0.00
V9	-0.04	-0.00	-0.01	0.01
V10	-0.04	-0.00	-0.00	0.02
duplicate1		3.37		1.10

Table 3: Variable importance scores for part (c) simulation.

```
> bagImp1$Variable <- factor(rownames(bagImp1), levels = vnames)
>
> bagImp2 <- varImp(bagFit2)
> names(bagImp2) <- "Extra"
> bagImp2$Variable <- factor(rownames(bagImp2), levels = vnames)
>
> bagImps <- merge(bagImp1, bagImp2, all = TRUE)
> rownames(bagImps) <- bagImps$Variable
> bagImps$Variable <- NULL
```

Table 4 indicates that predictors `V1`–`V5` are at the top of the importance ranking. However `V6`–`V10` have relatively higher importance scores as compared to the random forest importance scores. Adding an extra highly correlated predictor with `V1` has less of an impact on the overall importance score for `V1` as compared to random forest.

For Cubist, Table **??** indicates that predictors `V1`–`V5` are at the top of the importance ranking. Adding an extra highly correlated predictor with `V1` has very little impact on the importance scores when using Cubist.

```
> library(Cubist)
> set.seed(147)
> cbFit1 <- cubist(x = simulated[, 1:10],
+                  y = simulated$y,
+                  committees = 100)
> cbImp1 <- varImp(cbFit1)
> names(cbImp1) <- "Original"
> cbImp1$Variable <- factor(rownames(cbImp1), levels = vnames)
>
> set.seed(147)
> cbFit2 <- cubist(x = simulated[, names(simulated) != "y"],
+                  y = simulated$y,
```

	Original	Extra
V1	1.92	1.81
V2	2.30	2.31
V3	1.37	1.21
V4	2.77	2.68
V5	2.43	2.30
V6	1.00	0.87
V7	0.94	0.88
V8	0.58	0.49
V9	0.68	0.59
V10	0.86	0.80
duplicate1		1.46

Table 4: Variable importance scores for part (d) simulation using bagging.

```
+                  committees = 100)
> cbImp2 <- varImp(cbFit2)
> names(cbImp2) <- "Extra"
> cbImp2$Variable <- factor(rownames(cbImp2), levels = vnames)
>
> cbImp <- merge(cbImp1, cbImp2, all = TRUE)
> rownames(cbImp) <- cbImp$Variable
> cbImp$Variable <- NULL
```

	Original	Extra
V1	71.50	70.50
V2	58.50	58.00
V3	47.00	44.50
V4	48.00	48.00
V5	33.00	32.50
V6	13.00	13.50
V7	0.00	0.00
V8	0.00	0.00
V9	0.00	0.00
V10	0.00	0.00
duplicate1		1.00

Table 5: Variable importance scores for part (d) simulation using Cubist.

Exercise 2

Use a simulation to show tree bias with different granularities.

Solutions

Recall (?) found that predictors that are more granular (or have more potential split points) have a greater chance of being used towards the top of a tree to partition, even if the predictor has little-to-no relationship with the response. To investigate this phenomenon, let's develop a simple simulation. For the simulation, we will generate one categorical predictor that is informative at separating the response into two more homogenous groups. We will also generate a continuous predictor that is not informative at separating the response into two more homogenous groups. We will then use both of these predictors to build a one-split tree and note which predictor is used to split the data. This simulation will be run many times and we will tally the number of times each predictor is used as the first split.

```
> set.seed(102)
> X1 <- rep(1:2,each=100)
> Y <- X1 + rnorm(200,mean=0,sd=4)
> set.seed(103)
> X2 <- rnorm(200,mean=0,sd=2)
>
> simData <- data.frame(Y=Y,X1=X1,X2=X2)
```

The code chuck above defines how each predictor (X1 and X2) are related to the response. Predictor X1 has two categories and is created to separate the response into two more homogenous groups. Predictor X2, however, is not related to the response. Figure 1 illustrates the relationship between each predictor and the response.

In this simulation, the frequency that each predictor is selected in presented in Table 5. In this case, X1 and X2 are selected in near equal proportions despite the fact that the response is defined based on information from X1. As the amount of noise in the simulation increases, the chances that X2 are selected increase. Conversely, as the amount of noise decreases the chance that X2 is selected decreases. This implies that the granularity provided by X2 has a strong influence on whether or not it is selected–not the fact that it has no association with the response.

X2	52
X1	44

Table 6: Frequency of predictor selection for tree bias simulation.

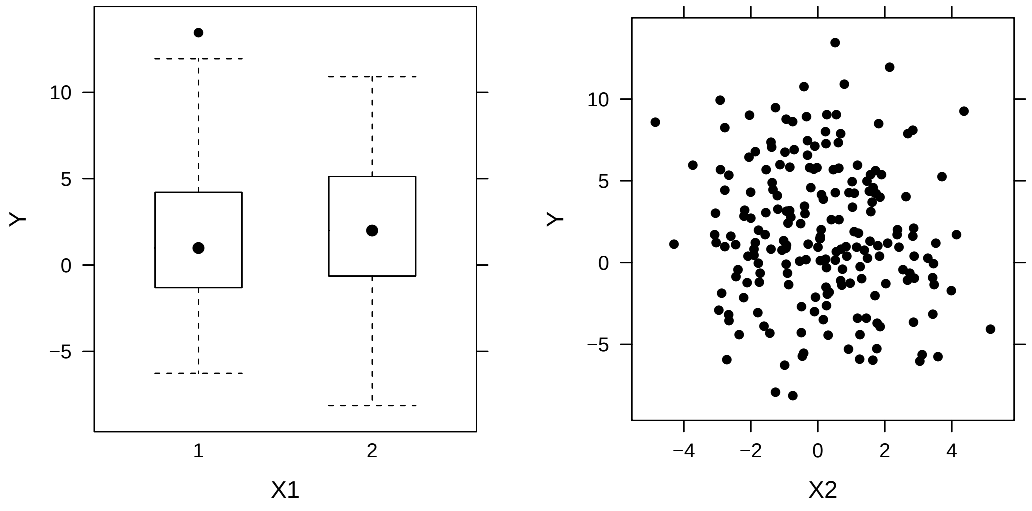

Figure 1: The univariate relationship with each predictor and the response. Predictor X1 has only two categories, but is defined to to create two more homogenous groups with respect to the response. Predictor X2 has 200 possible categories (is more granular) and is not related to the response.

Exercise 3

In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by the gradient. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 2 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data. The left-hand plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9.

(a) Why does the model on the right focus its importance on just the first few of predictors, whereas the model on the left spreads importance across more predictors?

(b) Which model do you think would be more predictive of other samples?

(c) How would increasing interaction depth affect the slope of predictor importance for either model in Figure 2?

Solutions

The model on the right focuses importance on just a few predictors for a couple of reasons. First, as the learning rate increases towards 1, the model becomes more greedy. As greediness increases, the model will be more likely to identify fewer predictors related to the response. Second, as the bagging fraction increases, the model uses more of the data in model construction. The less the stochastic element of the method (i.e. larger bagging fraction) the fewer predictors will be identified as important. Therefore, as the learning rate and bagging fraction increase, the importance will be concentrated on fewer and fewer predictors.

At the same time, as the values of these parameters increase, model performance will correspondingly decrease. Hence, the model on the left is likley to have better performance than the model on the right.

Interaction depth also relatively affects the variable importance metric. As tree depth increases, variable importance is likely to be spread over more predictors increasing the length of the horizontal lines in the importance figure.

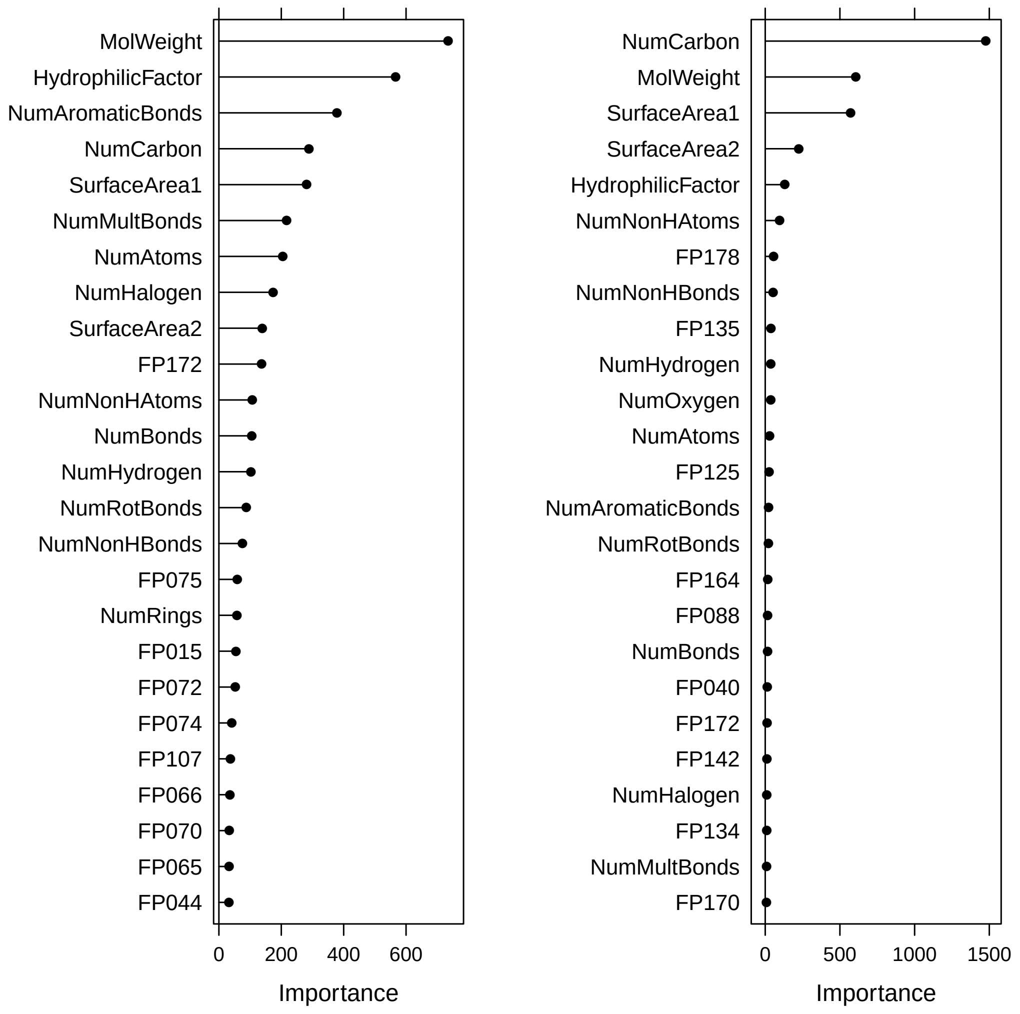

Figure 2: A comparison of variable importance magnitudes for differing values of the bagging fraction and shrinkage parameters. Both tuning parameters are set to 0.1 in the left figure. Both are set to 0.9 in the right figure.

Exercise 4

Use a single predictor in the solubility data, such as the molecular weight or the number of carbon atoms, fit several models:

(a) a simple regression tree

(b) a random forest model

(c) different Cubist models with: a single rule or multiple committees (each with and without using neighbor adjustments).

Using the test set data, plot the predictor data versus the solubility results. Overlay the model predictions for the test set. How do the model differ? Does changing the tuning parameter(s) significantly affect the model fit?

Solutions

The code listed below constructs models from Parts (a) through (c), and the performance results of these models are provided in Table 6. Not surprisingly, the single tree performs the worst. The randomness and iterative process incorporated using Random Forest improves predictive ability when using just this one predictor. For the Cubist models, a couple of trends can be seen. First, the no neighbor models perform better than the corresponding models that were tuned using multiple neighbors. At the same time, using multiple committees slightly improves the predictive ability of the models. Still, the best Cubist model (multiple committees and no neighbors) performs slightly worse than the random forest model.

```
> data(solubility)
>
> solTrainMW <- subset(solTrainXtrans,select="MolWeight")
> solTestMW <- subset(solTestXtrans,select="MolWeight")
>
> set.seed(100)
> rpartTune <- train(solTrainMW, solTrainY,
+                    method = "rpart2",
+                    tuneLength = 1)
> rpartTest <- data.frame(Method = "RPart",Y=solTestY,
+                         X=predict(rpartTune,solTestMW))
>
> rfTune <- train(solTrainMW, solTrainY,
+                 method = "rf",
+                 tuneLength = 1)
> rfTest <- data.frame(Method = "RF",Y=solTestY,
+                      X=predict(rfTune,solTestMW))
>
```

```
>
> cubistTune1.0 <- train(solTrainMW, solTrainY,
+                        method = "cubist",
+                        verbose = FALSE,
+                        metric = "Rsquared",
+                        tuneGrid = expand.grid(committees = 1,
+                                               neighbors = 0))
> cubistTest1.0 <- data.frame(Method = "Cubist1.0",Y=solTestY,
+                             X=predict(cubistTune1.0,solTestMW))
>
> cubistTune1.n <- train(solTrainMW, solTrainY,
+                        method = "cubist",
+                        verbose = FALSE,
+                        metric = "Rsquared",
+                        tuneGrid = expand.grid(committees = 1,
+                                               neighbors = c(1,3,5,7)))
> cubistTest1.n <- data.frame(Method = "Cubist1.n",Y=solTestY,
+                             X=predict(cubistTune1.n,solTestMW))
>
> cubistTune100.0 <- train(solTrainMW, solTrainY,
+                         method = "cubist",
+                         verbose = FALSE,
+                         metric = "Rsquared",
+                         tuneGrid = expand.grid(committees = 100,
+                                                neighbors = 0))
> cubistTest100.0 <- data.frame(Method = "Cubist100.0",Y=solTestY,
+                               X=predict(cubistTune100.0,solTestMW))
>
> cubistTune100.n <- train(solTrainMW, solTrainY,
+                         method = "cubist",
+                         verbose = FALSE,
+                         metric = "Rsquared",
+                         tuneGrid = expand.grid(committees = 100,
+                                                neighbors = c(1,3,5,7)))
> cubistTest100.n <- data.frame(Method = "Cubist100.n",Y=solTestY,
+                               X=predict(cubistTune100.n,solTestMW))
```

Method	R2
Recursive Partitioning	0.33
Random Forest	0.48
Cubist.SingleRule.NoNeighbors	0.42
Cubist.SingleRule.MultNeighbors	0.38
Cubist.MultCommittees.NoNeighbors	0.45
Cubist.MultCommittees.MultNeighbors	0.39

Table 7: Model performance using only Molecular Weight as a predictor.

Test set performance is illustrated in Figure 3. The performance for recursive partitioning stands out since there are only two possible X values due to the split on the single predictor. Performance across random forest and the Cubist models are similar, with random forest having slightly smaller vertical spread across the range of the line of agreement. All of the Cubist models appear to have a lower-bound on predicted values at approximately -4.5.

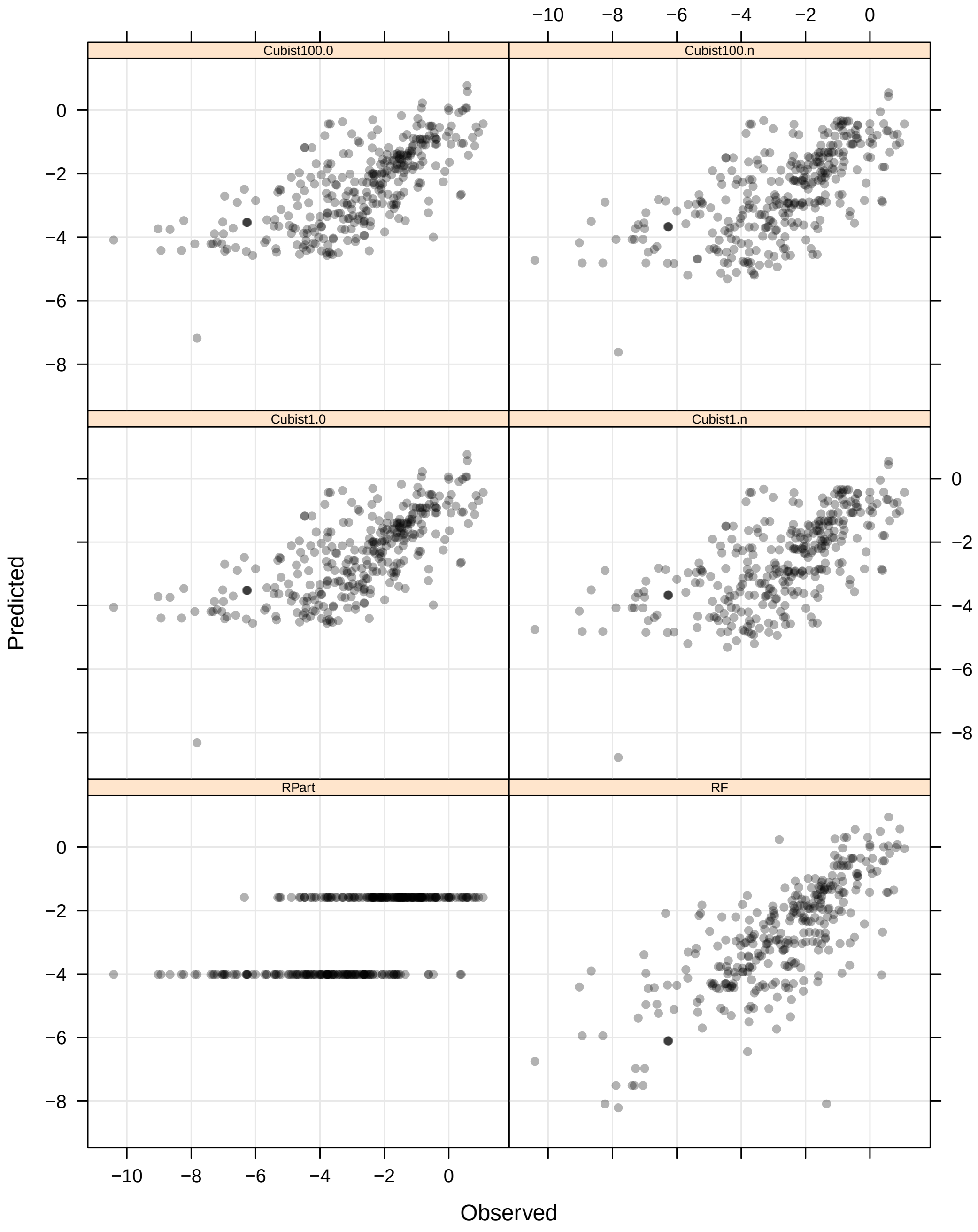

Figure 3: Test set performance across models using only Molecular Weight as a predictor.

Exercise 5

Fit different tree– and rule–based models for the Tecator data discussed in Exercise 6.1. How do they compare to linear models? Do the between–predictor correlations seem to affect your models? If so, how would you transform or re–encode the predictor data to mitigate this issue?

Solutions

The optimal RMSE in Exercise 6.1 came from the PLS model and was 0.65. We can load the data in the same way as before:

```
> library(caret)
> data(tecator)
>
> set.seed(1029)
> inMeatTraining <- createDataPartition(endpoints[, 3], p = 3/4, list= FALSE)
>
> absorpTrain <- absorp[ inMeatTraining,]
> absorpTest  <- absorp[-inMeatTraining,]
> proteinTrain <- endpoints[ inMeatTraining, 3]
> proteinTest  <- endpoints[-inMeatTraining,3]
>
> absorpTrain <- as.data.frame(absorpTrain)
> absorpTest  <- as.data.frame(absorpTest)
>
> ctrl <- trainControl(method = "repeatedcv", repeats = 5)
```

A simple CART model can be fit using the syntax here:

```
> set.seed(529)
> meatCART <- train(x = absorpTrain, y = proteinTrain,
+                   method = "rpart",
+                   trControl = ctrl,
+                   tuneLength = 25)
```

The resulting tuning parameter profile is presented in Figure 4. For this mode, the optimal RMSE is 2.635. This value is worse than the optimal value found using the PLS model.

Next we will tune and evaluate the following models: bagged trees, random forest, gradient boosting machines, and Cubist. The tuning parameter profiles for random forest, gradient boosting machines, and Cubist can be found in Figures 5, 6, and 7, respectively.

```
> set.seed(529)
> meatBagged <- train(x = absorpTrain, y = proteinTrain,
+                     method = "treebag",
+                     trControl = ctrl)
```

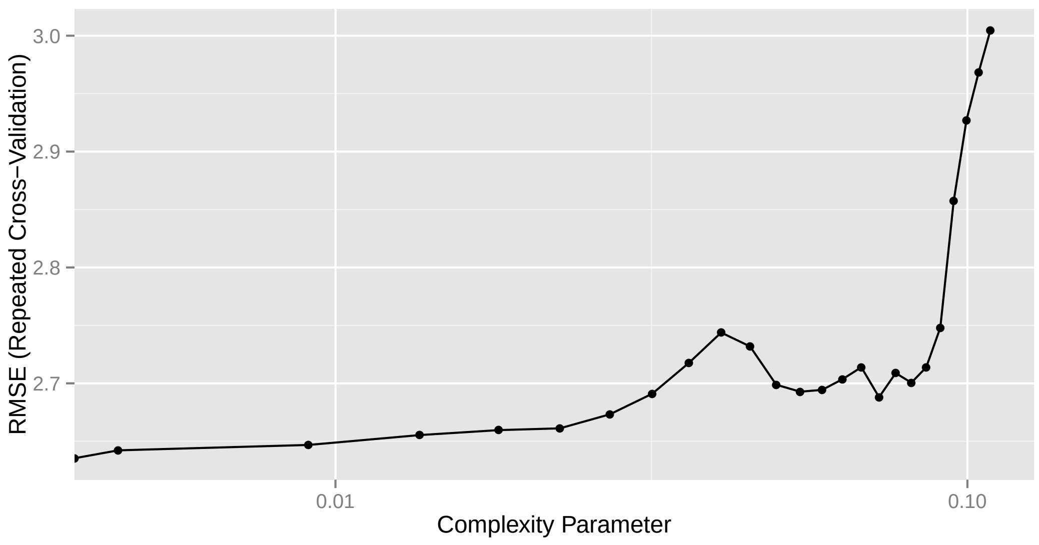

Figure 4: The RMSE resampling profile for the single CART model.

```
> set.seed(529)
> meatRF <- train(x = absorpTrain, y = proteinTrain,
+                 method = "rf",
+                 ntree = 1500,
+                 tuneLength = 10,
+                 trControl = ctrl)
```

```
> gbmGrid <- expand.grid(interaction.depth = seq(1, 7, by = 2),
+                        n.trees = seq(100, 1000, by = 50),
+                        shrinkage = c(0.01, 0.1))
> set.seed(529)
> meatGBM <- train(x = absorpTrain, y = proteinTrain,
+                  method = "gbm",
+                  verbose = FALSE,
+                  tuneGrid = gbmGrid,
+                  trControl = ctrl)
```

```
> set.seed(529)
> meatCubist <- train(x = absorpTrain, y = proteinTrain,
+                     method = "cubist",
+                     verbose = FALSE,
+                     tuneGrid = expand.grid(committees = c(1:10, 20, 50, 75, 100),
+                                            neighbors = c(0, 1, 5, 9)),
+                     trControl = ctrl)
```

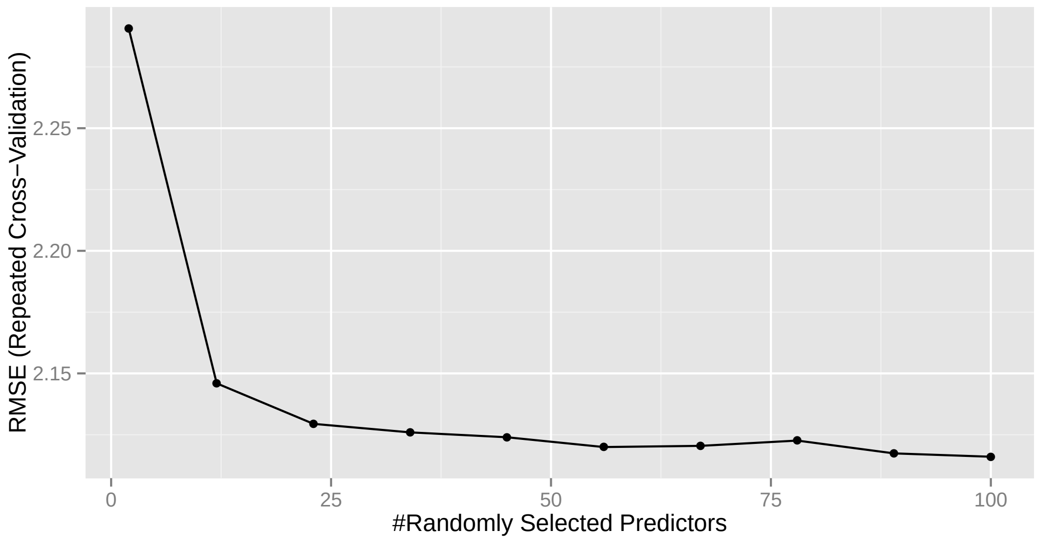

Figure 5: The RMSE resampling profile for the random forest model.

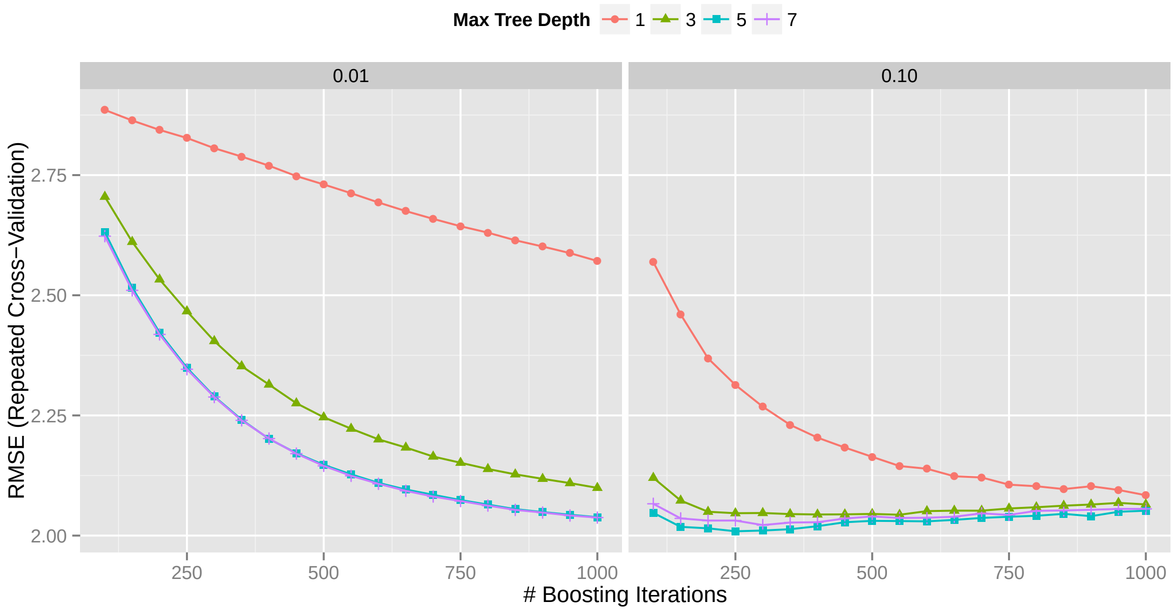

Figure 6: The RMSE resampling profile for the gradient boosting machine model.

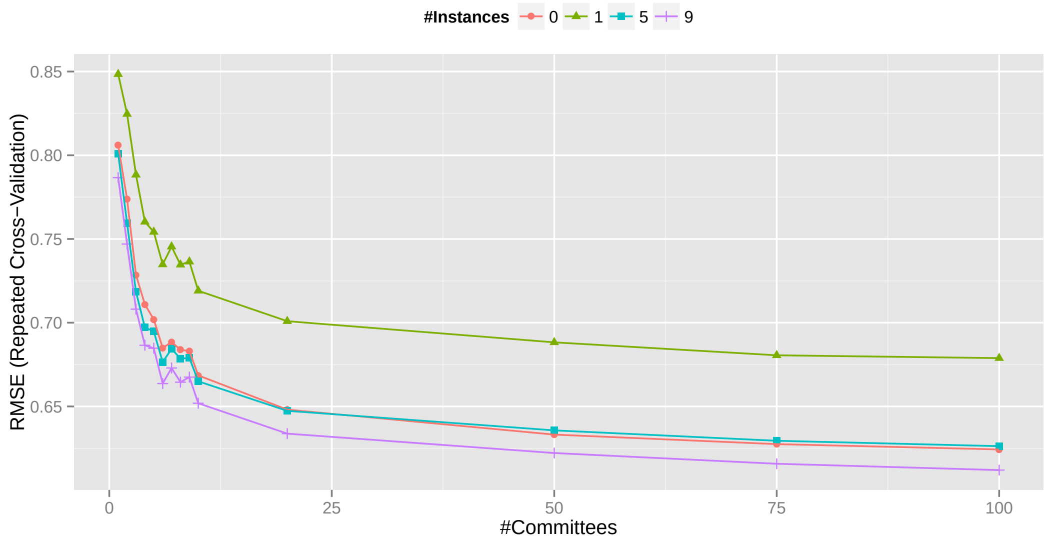

Figure 7: The RMSE resampling profile for the cubist model.

```
> load("meatPLS.RData")
> load("meatNet.RData")
> meatResamples <- resamples(list(CART = meatCART,
+                                 GBM = meatGBM,
+                                 Cubist = meatCubist,
+                                 "Bagged Tree" = meatBagged,
+                                 "Random Forest" = meatRF,
+                                 PLS = meatPLS,
+                                 "Neural Network" = meatNet))
```

To compare model performance across those built in Chapters 6, 7, and 8, we can examine the resampling performance distributions (Figure 8). Clearly the distributions of the PLS, Cubist, and neural network models indicate better performance than the tree-based models with RMSE values well under 1 and less overall variation.

The latent variable characteristic of PLS and neural network models could be crucial model characteristics for this data and could be better suited for handling between-predictor correlations.

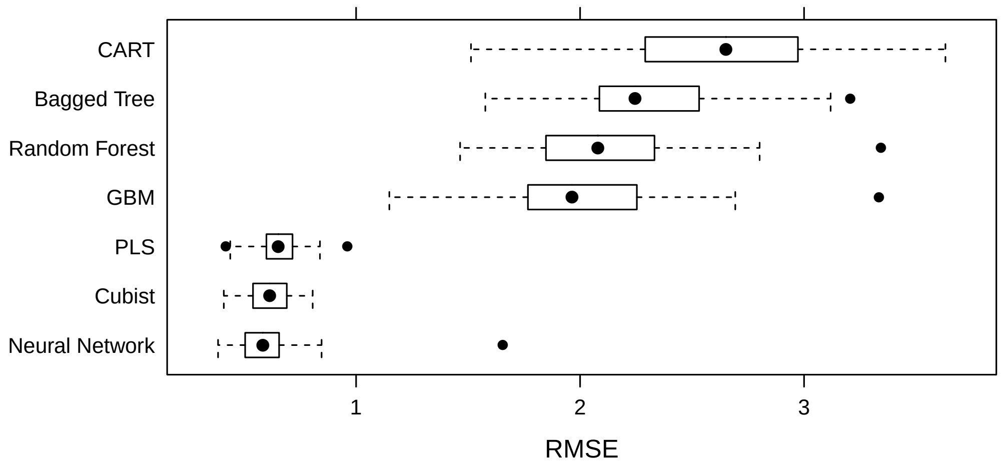

Figure 8: Resampling distributions of tree– and rule–based models, along with the best models from the previous two chapters (PLS and neural networks).

Exercise 6

Return to the permeability problem described in Exercises 6.2 and 7.4. Train several tree–based models and evaluate the resampling and test set performance.

(a) Which tree–based model gives the optimal resampling and test set performance?

(b) Do any of these models outperform the linear or non–linear based regression models you have previously developed for this data? What criteria did you use to compare models' performance?

(c) Of all the models you have developed thus far, which, if any, would you recommend to replace the permeability laboratory experiment?

Solutions

In order to make a parallel comparison to the results in Exercises 6.2 and 7.4, we need to perform the same pre-processing steps and set up the identical validation approach. Recall that the optimal R^2 value for linear based methods was 0.58 (Elastic Net) and for non-linear based methods was 0.55 (SVM). The following syntax provides the same pre-processing, data partition into training and testing sets, and validation set-up.

```
> library(AppliedPredictiveModeling)
> data(permeability)
>
> #Identify and remove NZV predictors
> nzvFingerprints <- nearZeroVar(fingerprints)
> noNzvFingerprints <- fingerprints[,-nzvFingerprints]
>
> #Split data into training and test sets
> set.seed(614)
> trainingRows <- createDataPartition(permeability,
+                                     p = 0.75,
+                                     list = FALSE)
>
> trainFingerprints <- noNzvFingerprints[trainingRows,]
> trainPermeability <- permeability[trainingRows,]
>
> testFingerprints <- noNzvFingerprints[-trainingRows,]
> testPermeability <- permeability[-trainingRows,]
>
> set.seed(614)
> ctrl <- trainControl(method = "LGOCV")
```

Next, we will find optimal tuning parameters for simple CART, RF and GBM models.

```
> set.seed(614)
> rpartGrid <- expand.grid(maxdepth= seq(1,10,by=1))
> rpartPermTune <- train(x = trainFingerprints, y = log10(trainPermeability),
+                         method = "rpart2",
+                         tuneGrid = rpartGrid,
+                         trControl = ctrl)
```

```
> set.seed(614)
>
> rfPermTune <- train(x = trainFingerprints, y = log10(trainPermeability),
+                      method = "rf",
+                      tuneLength = 10,
+                      importance = TRUE,
+                      trControl = ctrl)
```

```
> set.seed(614)
> gbmGrid <- expand.grid(interaction.depth=seq(1,6,by=1),
+                         n.trees=c(25,50,100,200),
+                         shrinkage=c(0.01,0.05,0.1))
> gbmPermTune <- train(x = trainFingerprints, y = log10(trainPermeability),
+                       method = "gbm",
+                       verbose = FALSE,
+                       tuneGrid = gbmGrid,
+                       trControl = ctrl)
```

Figure 9 indicates that the optimal tree depth that maximizes R^2 is 4, with an R^2 of 0.62. This result is slightly better than what we found with either the selected linear or non-linear based methods.

Figure 10 indicates that the optimal m_{try} value that maximizes R^2 is 388, with an R^2 of 0.63. The tuning parameter profile as well as the similar performance results with recursive partitioning indicates that the underlying data structure is fairly consistent across the samples. Hence, the modeling process does not benefit from the reduction in variance induced by random forests.

Next, let's look at the variable importance of the top 10 predictors for the random forest model (Figure 11). Clearly a handful of predictors are identified as most important by random forests.

Figure 12 indicates that the optimal interaction depth, number of trees, and shrinkage that maximize R^2 are 4, 200, and 0.05, respectively, with an R^2 of 0.59.

There are a couple of interesting characteristics we see from the GBM tuning parameter profiles. First, fewer trees with a tiny amount of shrinkage is optimal. This, again, points to the stability of the underlying samples. Second, a more complex model like GBM is not necessary for this data. Instead, a simpler model like a linear-based technique or a single CART tree provides near optimal results while at the same time being more interpretable than, say, the optimal random forest model.

The optimal recursive partitioning tree is presented in Figure 13. This tree reveals that similar to the variable importance rankings from random forests, X6, X93, and X157 play an important

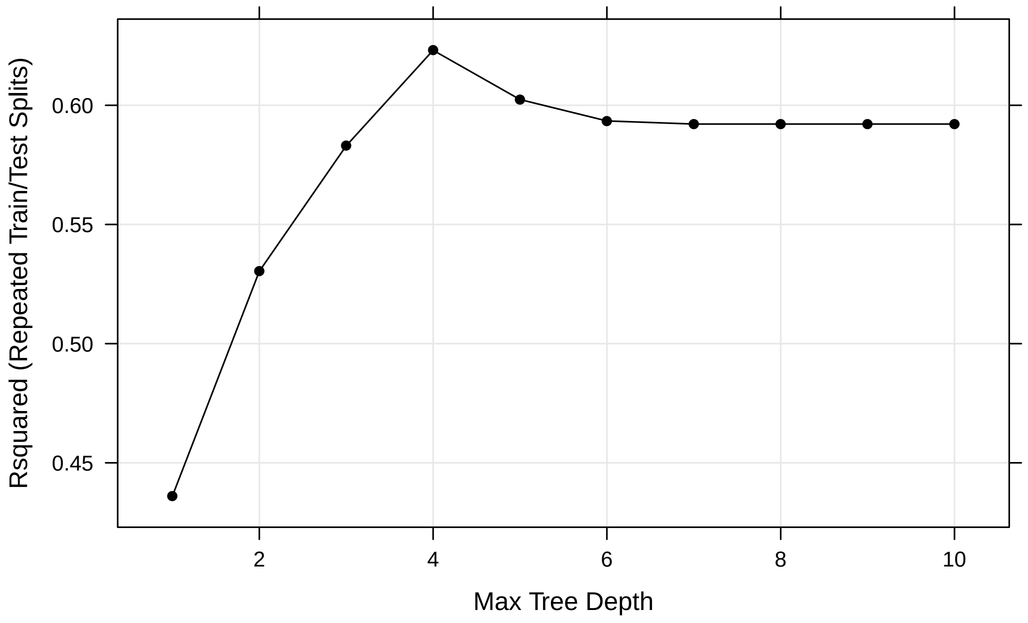

Figure 9: Recursive partitioning tuning parameter profile for the permeability data

role in separating samples. Also, the splits reveal the impact of the presence (>0.5) or absence of the fingerprint on permeability. Having fingerprint X6 appears to be associated with higher overall permeability values. Likewise, not having fingerprint X6 while having fingerprint X93 appears to be associated with lower overall permeability values.

The findings of this exercise as well as 6.2 and 7.4 indicate that an interpretable model like recursive partitioning ($R^2 = 0.62$) performs just as well as any of the more complex models. An R^2 at this level may or may not be sufficient to replace the permeability laboratory experiment. However, these findings may enable a gross computational screening which could identify compounds that are likely to be at the extremes of permeability. The predictors identified by recursive partitioning and random forests may also provide key insights about structures that are relevant to compounds' permeability.

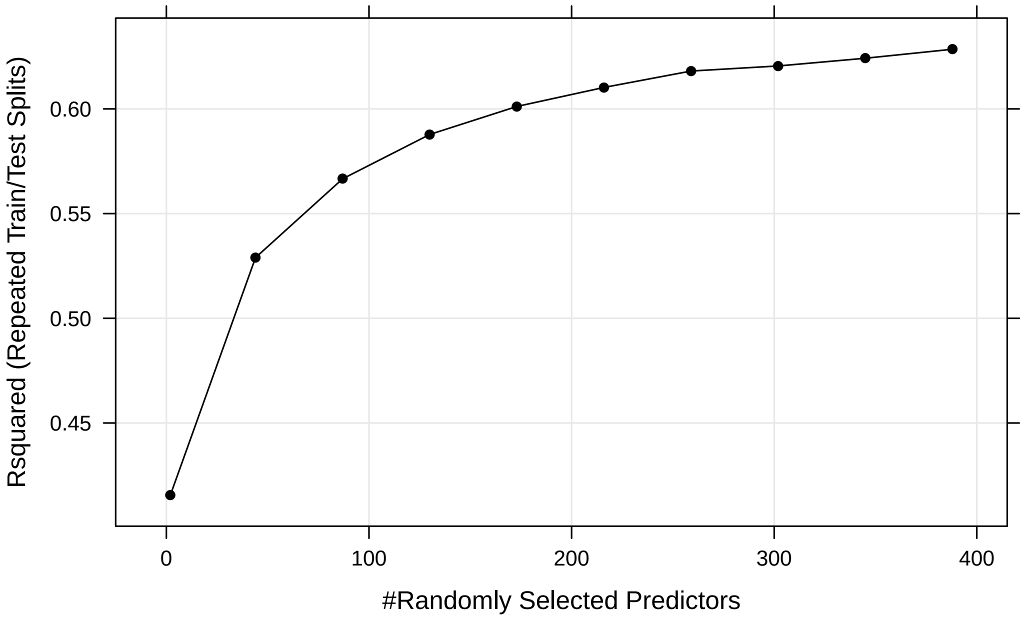

Figure 10: Random forest tuning parameter profile for the permeability data

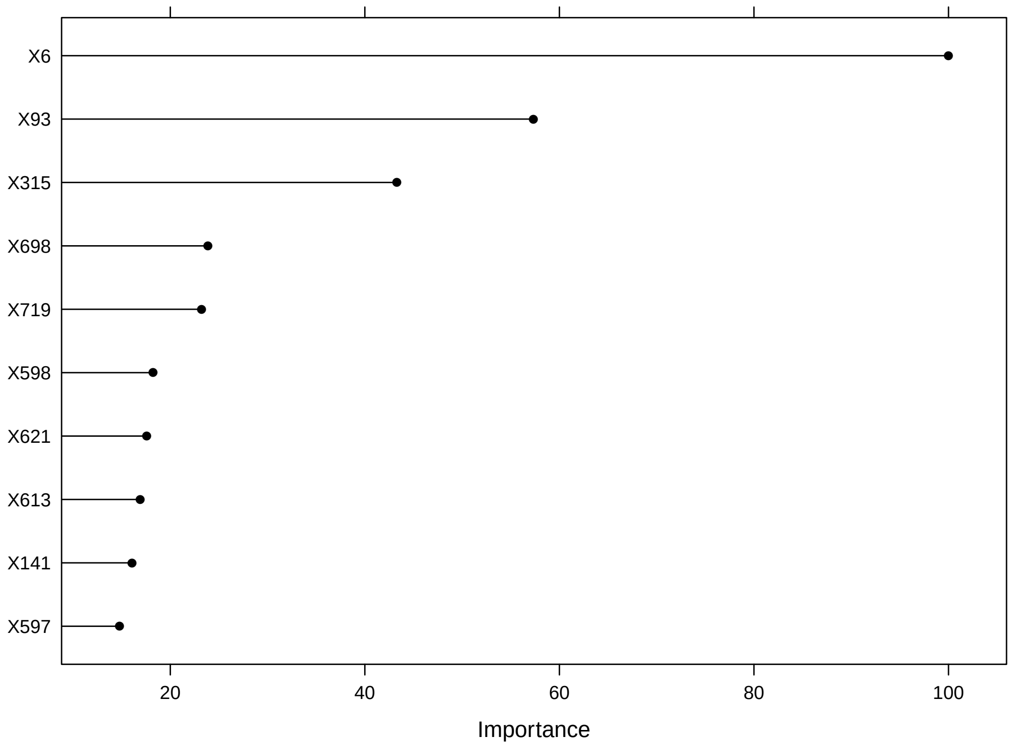

Figure 11: Variable importance for RF model for permeability data

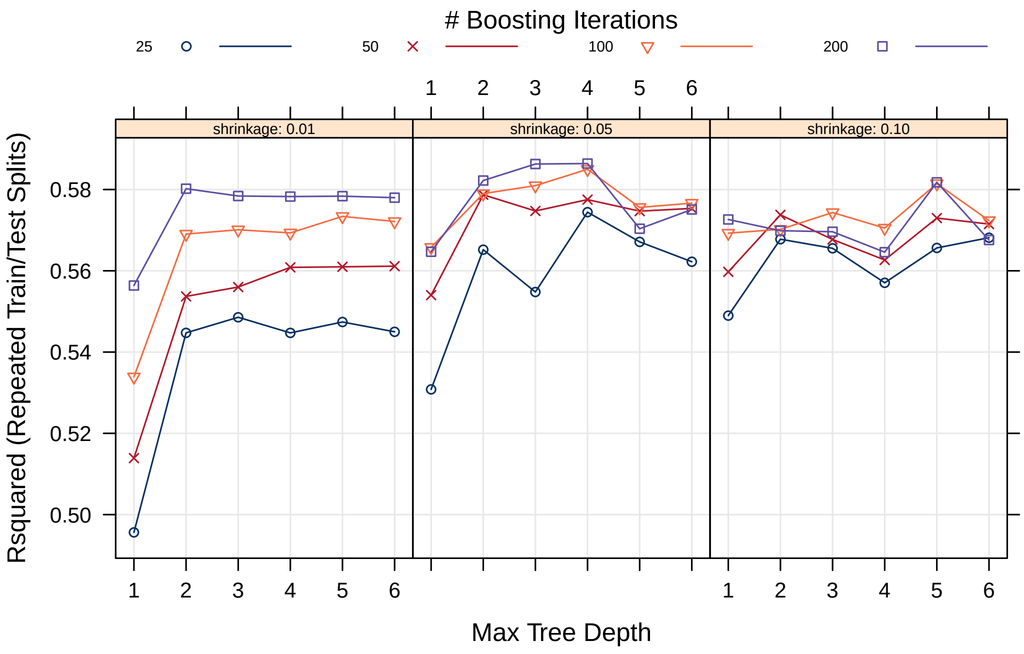

Figure 12: Gradient boosting machine tuning parameter profile for the permeability data

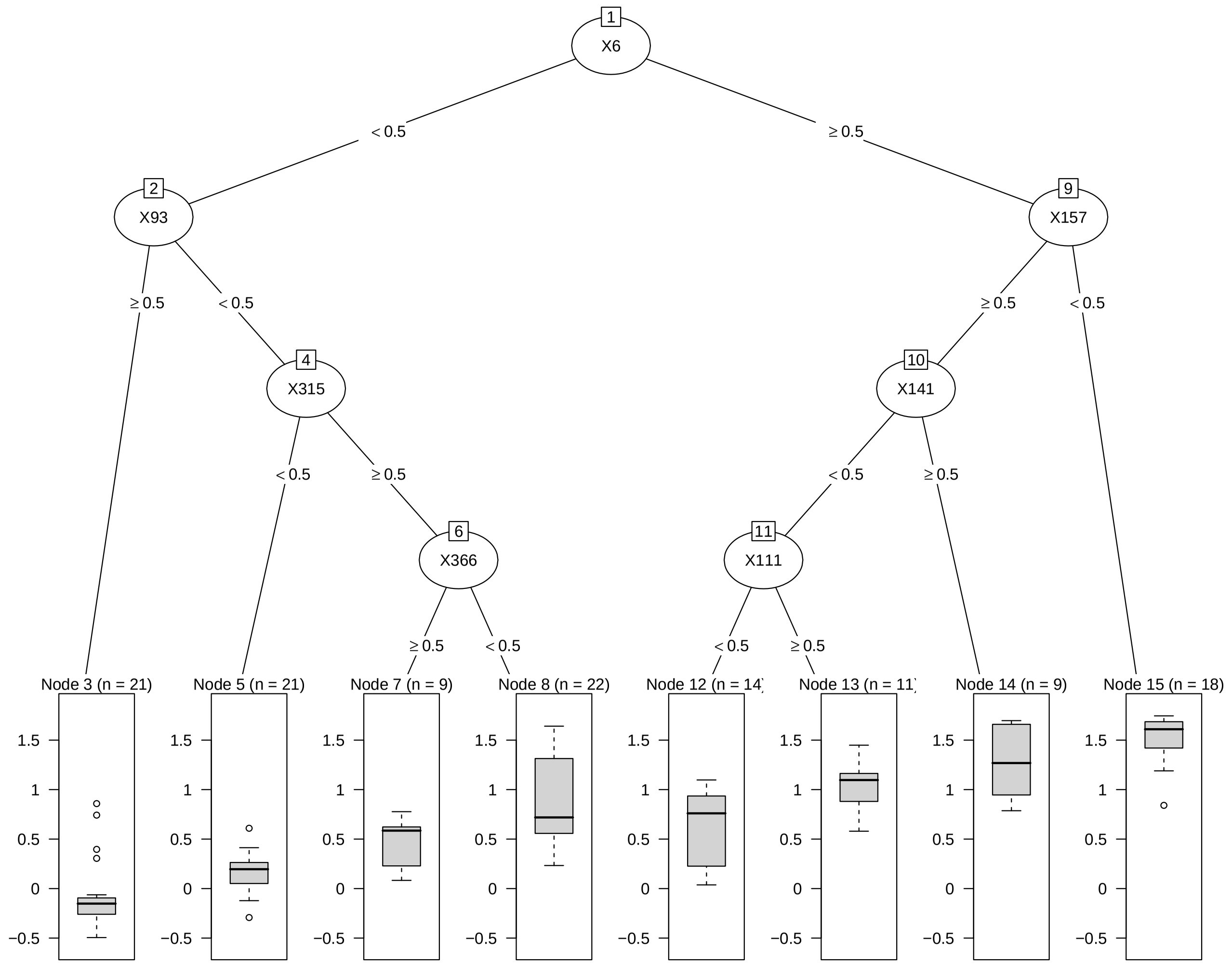

Figure 13: Optimal recursive partitioning tree for permeability data

Exercise 7

Refer to Exercises 6.3 and 7.5 which describe a chemical manufacturing process. Use the same data imputation, data–splitting and pre–processing steps as before and train several tree–based models.

(a) Which tree–based regression model gives the optimal resampling and test set performance?

(b) Which predictors are most important in the optimal tree–based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and non–linear models?

(c) Plot the optimal single tree with the distribution of yield in the terminal nodes. Does this view of the data provide additional knowledge about the biological or process predictors and their relationship with yield?

Solutions

We will use the same pre-processing steps and validation approach as in Exercises 6.3 and 7.5. In Exercise 6.3, the cross-validated R^2 value for PLS was 0.57, and the key predictors were manufacturing processes 32, 09, and 13. The pairwise plots of these predictors with the response may indicate a non–linear pattern of these predictors with the response. In Exercise 7.5, the MARS model was optimal with a cross-validated R^2 of 0.52. MARS singled out manufacturing processes 32 and 09 for predicting the response.

```
> library(AppliedPredictiveModeling)
> data(ChemicalManufacturingProcess)
>
> predictors <- subset(ChemicalManufacturingProcess,select= -Yield)
> yield <- subset(ChemicalManufacturingProcess,select="Yield")
>
> set.seed(517)
> trainingRows <- createDataPartition(yield$Yield,
+                                     p = 0.7,
+                                     list = FALSE)
>
> trainPredictors <- predictors[trainingRows,]
> trainYield <- yield[trainingRows,]
>
> testPredictors <- predictors[-trainingRows,]
> testYield <- yield[-trainingRows,]
>
> #Pre-process trainPredictors and apply to trainPredictors and testPredictors
> pp <- preProcess(trainPredictors,method=c("BoxCox","center","scale","knnImpute"))
> ppTrainPredictors <- predict(pp,trainPredictors)
> ppTestPredictors <- predict(pp,testPredictors)
>
```

```
> #Identify and remove NZV
> nzvpp <- nearZeroVar(ppTrainPredictors)
> ppTrainPredictors <- ppTrainPredictors[-nzvpp]
> ppTestPredictors <- ppTestPredictors[-nzvpp]
>
> #Identify and remove highly correlated predictors
> predcorr = cor(ppTrainPredictors)
> highCorrpp <- findCorrelation(predcorr)
> ppTrainPredictors <- ppTrainPredictors[, -highCorrpp]
> ppTestPredictors <- ppTestPredictors[, -highCorrpp]
>
> #Set-up trainControl
> set.seed(517)
> ctrl <- trainControl(method = "boot", number = 25)
```

Next, we will find optimal tuning parameters for simple CART, RF, GBM, and Cubist models.

```
> set.seed(614)
> rpartGrid <- expand.grid(maxdepth= seq(1,10,by=1))
> rpartChemTune <- train(x = ppTrainPredictors, y = trainYield,
+                         method = "rpart2",
+                         metric = "Rsquared",
+                         tuneGrid = rpartGrid,
+                         trControl = ctrl)
```

```
> set.seed(614)
>
> rfGrid <- expand.grid(mtry=seq(2,38,by=3))
>
> rfChemTune <- train(x = ppTrainPredictors, y = trainYield,
+                     method = "rf",
+                     tuneGrid = rfGrid,
+                     metric = "Rsquared",
+                     importance = TRUE,
+                     trControl = ctrl)
```

```
> set.seed(614)
> gbmGrid <- expand.grid(interaction.depth=seq(1,6,by=1),
+                         n.trees=c(25,50,100,200),
+                         shrinkage=c(0.01,0.05,0.1,0.2))
>
> gbmChemTune <- train(x = ppTrainPredictors, y = trainYield,
+                       method = "gbm",
+                       metric = "Rsquared",
+                       verbose = FALSE,
+                       tuneGrid = gbmGrid,
+                       trControl = ctrl)
```

```
> set.seed(614)
> cubistGrid <- expand.grid(committees = c(1, 5, 10, 20, 50, 100),
+                           neighbors = c(0, 1, 3, 5, 7))
>
> cubistChemTune <- train(x = ppTrainPredictors, y = trainYield,
+                         method = "cubist",
+                         verbose = FALSE,
+                         metric = "Rsquared",
+                         tuneGrid = cubistGrid,
+                         trControl = ctrl)
```

The optimal recursive partitioning model has a bootstrap CV R^2 value of 0.34. This result is substantially worse than what we found with either the optimal PLS or MARS models. The random forest and gradient boosting machine models have bootstrap CV R^2 values of 0.52 and 0.53, respectively. These are similar to the MARS model results but still not quite as good as the PLS model results.

Figure 14 provides the tuning parameter profile plot for the Cubist model. Here, the optimal bootstrap CV R^2 value is 0.62, which occurs using 1 neighbor and 100 committees. Therefore very localized information is useful in constructing the model; as the number the number of neighbors increases, the optimal performance decreases. Also adjusting the response via committees is necessary to improve prediction.

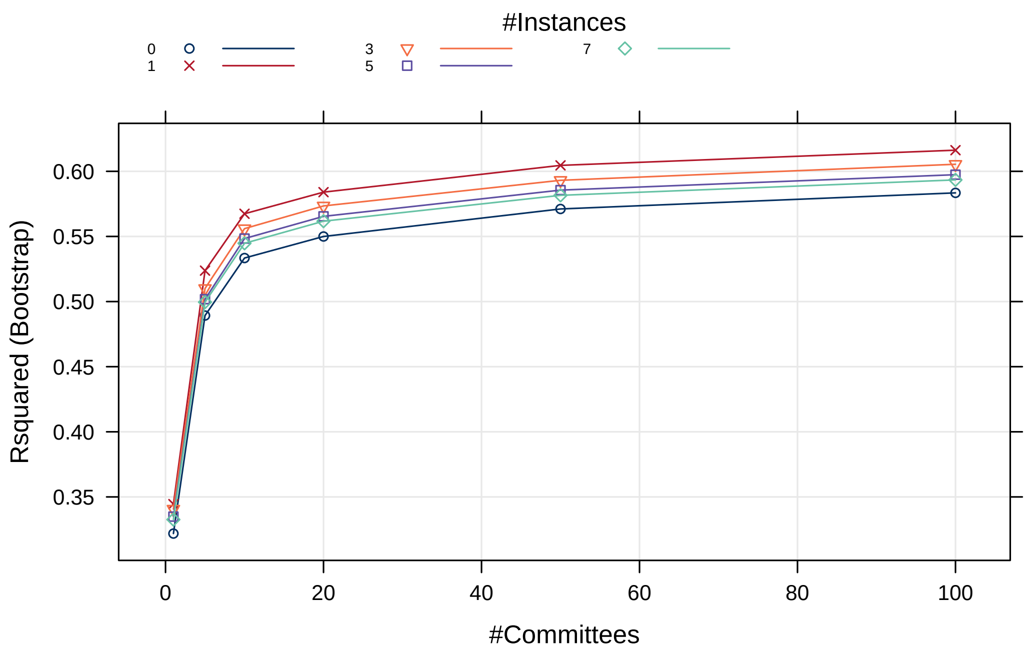

Figure 14: Cubist tuning parameter profile for the chemical manufacturing data

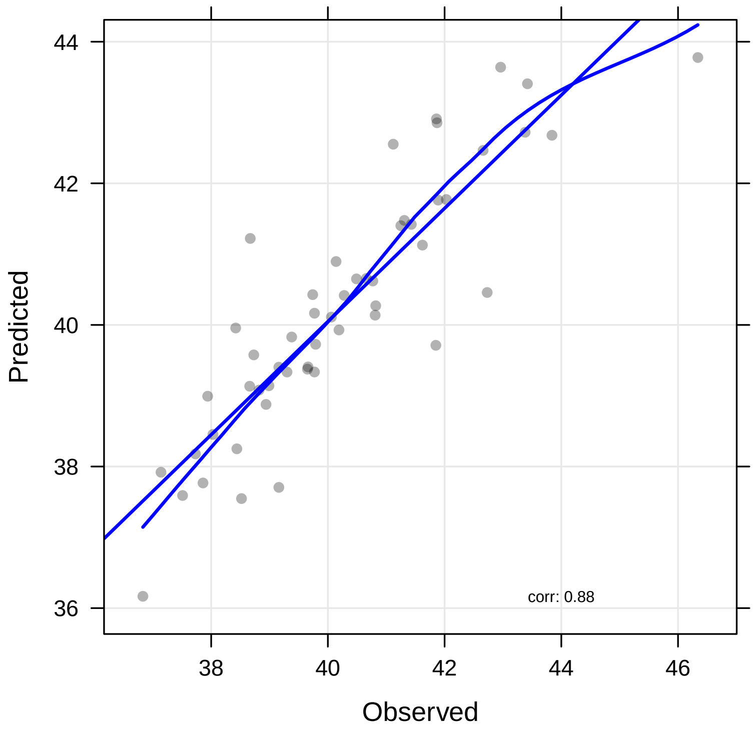

Figure 15: Cubist predictions for the test set for the chemical manufacturing data.

The Cubist model provides the best performance across all models we have tuned across this exercise as well as Exercises 6.3 and 7.5. The test set predictions for the Cubist model are presented in Figure 15, with an R^2 value of 0.782. This result is better than any of the previous models.

Figure 16 lists the top 15 important predictors for the manufacturing data. Again, manufacturing processes 32 and 09 are at the top, while process 13 is in the top 5. All models thus far point towards processes 32 and 09, which should be investigated further to better understand if these can be controlled to improve yield. The role of biological material 03 at its impact on yield should also be investigated.

The optimal recursive partitioning tree is presented in Figure 17. Manufacturing processes 32 and 13 are at the top, with higher values of process 32 being associated with larger yields. Lower values of process 32 are associated with smaller yields. However, a lower values of process 32 may be counter-acted with a corresponding lower value of process 13.

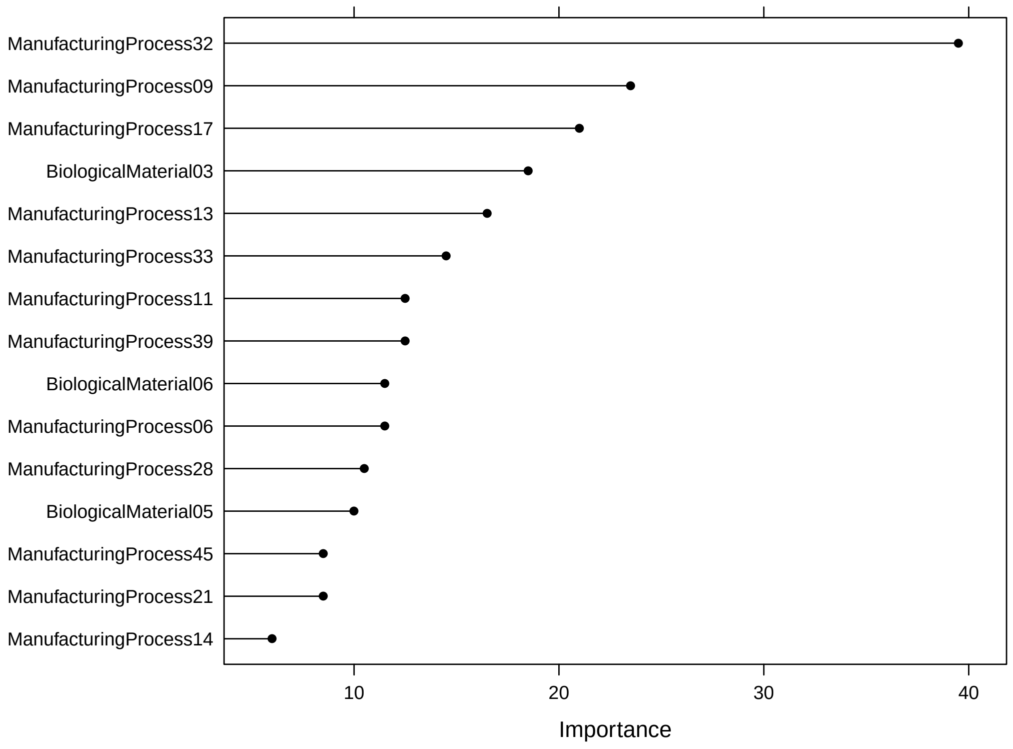

Figure 16: Cubist variable importance scores for the manufacturing data.

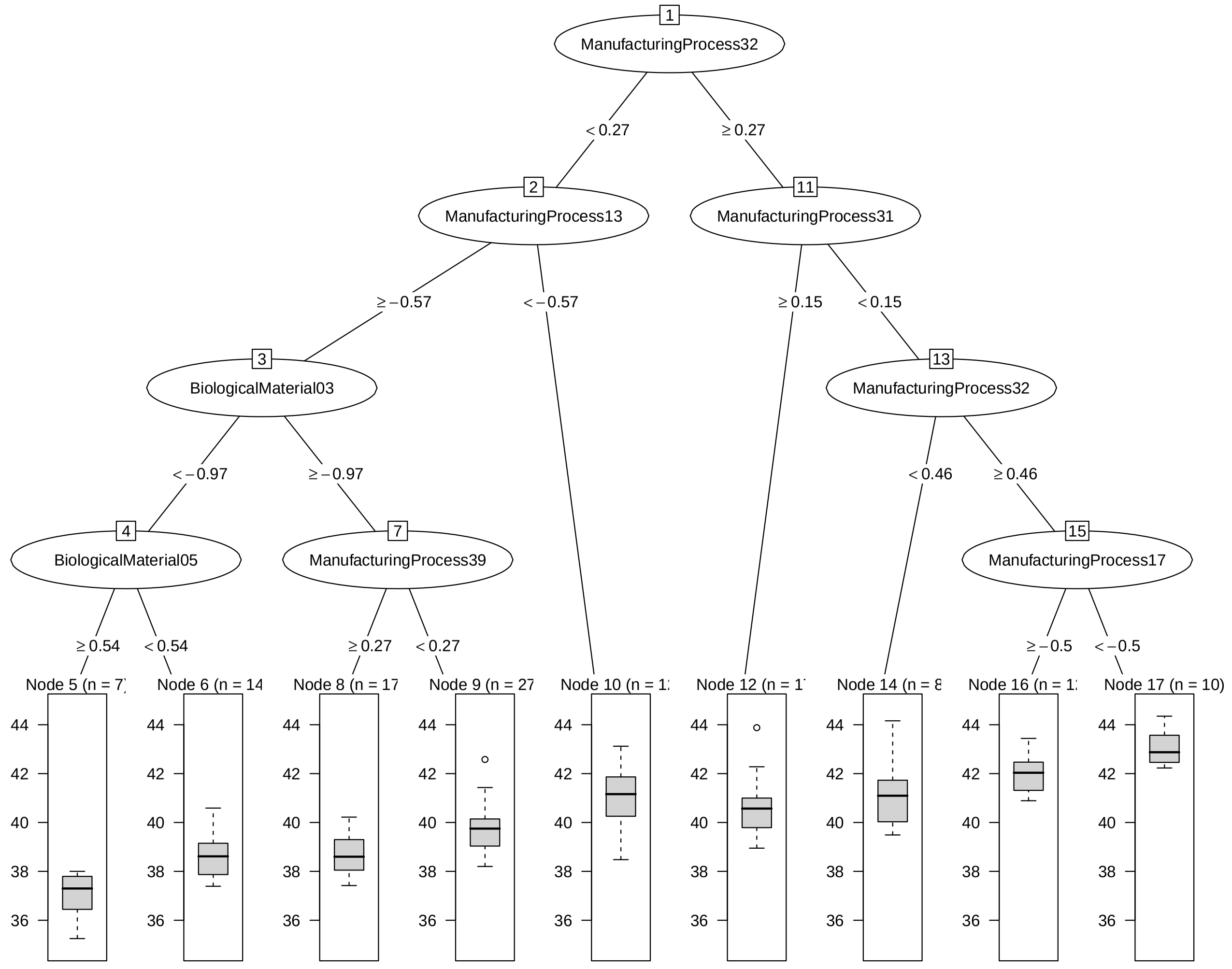

Figure 17: Optimal recursive partitioning tree for chemical manufacturing data

Session Info

- R Under development (unstable) (2014-12-29 r67265), `x86_64-apple-darwin10.8.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, parallel, splines, stats, stats4, utils
- Other packages: AppliedPredictiveModeling 1.1-6, caret 6.0-41, corrplot 0.73, Cubist 0.0.18, doMC 1.3.3, earth 4.1.0, elasticnet 1.1, foreach 1.4.2, gbm 2.1, ggplot2 1.0.0, ipred 0.9-3, iterators 1.0.7, kernlab 0.9-19, knitr 1.6, lars 1.2, lattice 0.20-29, latticeExtra 0.6-26, mlbench 2.1-1, modeltools 0.2-21, mvtnorm 1.0-2, nnet 7.3-8, party 1.0-19, partykit 0.8-0, plotmo 2.2.0, plotrix 3.5-7, pls 2.4-3, plyr 1.8.1, randomForest 4.6-10, RColorBrewer 1.0-5, reshape2 1.4, rpart 4.1-8, sandwich 2.3-1, strucchange 1.5-0, survival 2.37-7, xtable 1.7-3, zoo 1.7-11
- Loaded via a namespace (and not attached): BradleyTerry2 1.0-5, brglm 0.5-9, car 2.0-21, class 7.3-11, cluster 1.15.3, codetools 0.2-9, coin 1.0-24, colorspace 1.2-4, compiler 3.2.0, CORElearn 0.9.43, digest 0.6.4, e1071 1.6-3, evaluate 0.5.5, formatR 0.10, gtable 0.1.2, gtools 3.4.1, highr 0.3, labeling 0.2, lava 1.2.6, lme4 1.1-7, MASS 7.3-35, Matrix 1.1-4, minqa 1.2.3, munsell 0.4.2, nlme 3.1-118, nloptr 1.0.4, prodlim 1.4.3, proto 0.3-10, RANN 2.4.1, Rcpp 0.11.2, scales 0.2.4, stringr 0.6.2, tools 3.2.0