HW9

Team 4

02/11/2019

### 8.1

Recreate the simulated data from Exercise 7.2:

library(dplyr)  
library(tidyr)  
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"

1. Fit a random forest model to all of the predictors, then estimate the variable importance scores. Did the random forest model significantly use the uninformative predictors (V6 – V10)?

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

## Overall  
## V1 8.83890885  
## V2 6.49023056  
## V3 0.67583163  
## V4 7.58822553  
## V5 2.27426009  
## V6 0.17436781  
## V7 0.15136583  
## V8 -0.03078937  
## V9 -0.02989832  
## V10 -0.08529218

The random forest model does not significantly use the uninformative predictors (V6 – V10).

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

simulated\_dupl<-simulated  
simulated\_dupl$duplicate1 <- simulated\_dupl$V1 + rnorm(200) \* .1  
cor(simulated\_dupl$duplicate1, simulated\_dupl$V1)

## [1] 0.9396216

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

model2 <- randomForest(y ~ ., data = simulated\_dupl,  
 importance = TRUE,  
 ntree = 1000)  
rfImp2 <- varImp(model2, scale = FALSE)  
rfImp2

## Overall  
## V1 6.29780744  
## V2 6.08038134  
## V3 0.58410718  
## V4 6.93924427  
## V5 2.03104094  
## V6 0.07947642  
## V7 -0.02566414  
## V8 -0.11007435  
## V9 -0.08839463  
## V10 -0.00715093  
## duplicate1 3.56411581

Adding the predictor “duplicate 1” that is highly correlated with predictor V1 does change its importance score from 8.83890885 to 6.29780744 and duplicate1 got 3.56411581.  
Each node of each tree is constructed by finding single predictor and cutpoint for it. So only one candidate-predictor is examined at once.

1. Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in Strobl et al. (2007). Do these importances show the same pattern as the traditional random forest model?

library(party)  
model3 <- cforest(y ~., data = simulated\_dupl)  
varImp3 <- varimp(model3)  
varImp3

## V1 V2 V3 V4 V5   
## 6.571632689 6.110131511 0.010450332 7.485796796 1.889117623   
## V6 V7 V8 V9 V10   
## -0.006751589 0.007081381 -0.036022685 0.008976088 0.005356254   
## duplicate1   
## 2.722380892

The cforest function assigned slightly less weight to duplicate 1 and slightly more to V1, but overall pattern is pretty much similar to the pattern of the traditional random forest model.

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

Checking important variables with Cubist model

library(Cubist)  
library(dplyr)  
model4 <- cubist(y = simulated$y, x = simulated %>% select (-y), committees = 100)  
varImp4 <- varImp(model4)  
varImp4

## Overall  
## V1 71.5  
## V3 47.0  
## V2 58.5  
## V4 48.0  
## V5 33.0  
## V6 13.0  
## V7 0.0  
## V8 0.0  
## V9 0.0  
## V10 0.0

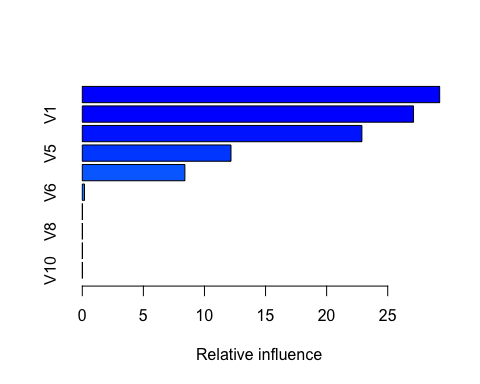
model4\_dupl <- cubist(y= simulated$y, x = simulated\_dupl%>% select (-y), committees = 100)  
varImp4 <- varImp(model4\_dupl)  
varImp4

## Overall  
## V1 64.5  
## V3 41.0  
## V2 60.0  
## V4 48.0  
## V5 31.0  
## V6 9.0  
## duplicate1 6.0  
## V8 2.0  
## V10 0.5  
## V7 0.0  
## V9 0.0

library(gbm)  
model5 <- gbm(y ~ ., data = simulated)

## Distribution not specified, assuming gaussian ...

summary(model5)

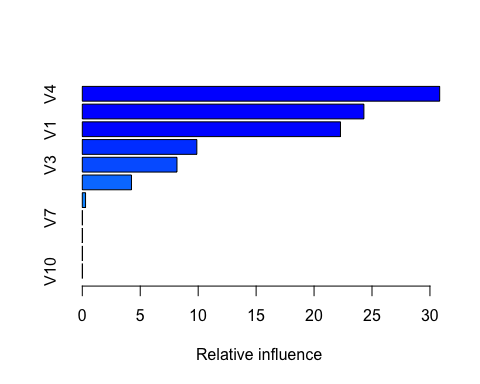


## var rel.inf  
## V4 V4 29.2533385  
## V1 V1 27.1118910  
## V2 V2 22.8860486  
## V5 V5 12.1706139  
## V3 V3 8.3972352  
## V6 V6 0.1808728  
## V7 V7 0.0000000  
## V8 V8 0.0000000  
## V9 V9 0.0000000  
## V10 V10 0.0000000

model5\_dupl<- gbm(y ~ ., data = simulated\_dupl)

## Distribution not specified, assuming gaussian ...

summary(model5\_dupl)



## var rel.inf  
## V4 V4 30.8404557  
## V2 V2 24.3006033  
## V1 V1 22.2798280  
## V5 V5 9.8840740  
## V3 V3 8.1702646  
## duplicate1 duplicate1 4.2443669  
## V6 V6 0.2804076  
## V7 V7 0.0000000  
## V8 V8 0.0000000  
## V9 V9 0.0000000  
## V10 V10 0.0000000

In Cubist and Gradient Boosting models duplicate variable has changed the importance score of V1 as that was with previous models, but relative importance of the duplicate variable is less in Cubist and Gradient Boosting models.

### 8.2

Use a simulation to show tree bias with different granularities.

The variable selection process in decision trees is biased, i.e., the predictor variable at a node of the tree might not actually be the predictor variable that is most important at this point. Predictors that have more potential split points (potentially noise variables) have a greater chance of being used towards the top of a tree to partition, even if the predictor has weak or no correlation with the response. In order to simulate and prove this we need to simulate one predictor that is informative and correlated with the target variable and one that is not (noise).

x\_inf is informative, high variance predictor that correlates to the target variable.

x\_noise is uninformative, low variance and not related to the response variable (noise).

y - target variable (x\_inf + pseudo-random variable)

rpart algorithm will be used to split the dataset recursively, which means that the subsets that arise from a split are further split until a predetermined termination criterion is reached. At each step, the split is made based on the independent variable that results in the largest possible reduction in heterogeneity of the dependent (predicted) variable.

library(rpart)  
set.seed(100)  
# assigning x\_inf variable  
x\_inf <- rep(1:2, each=500)  
# assigning target variable  
y <- x\_inf + rnorm(500, mean=0, sd=4)  
set.seed(100)  
# assigning noise  
x\_noise <- rnorm(500, mean = -4, sd = 2)  
# making data frame with created variables above  
simData <- data.frame(y=y, x\_inf=x\_inf, x\_noise=x\_noise)  
set.seed(100)  
  
fit <- rpart(y ~ ., data = simData)

# calculating important variables  
varImp(fit)

## Overall  
## x\_inf 0.5944102  
## x\_noise 4.3595405

Result indicates that uninformative variable was selected as the most important one confirming tree bias.

### 8.3

8.3. 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 8.24 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:

1. 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?

In general learning rate controls model greediness (the number of predictors), which means higher learning rate decreases the number of predictors. Stochastic Gradient Boosting creates and add trees to the model sequentially. New trees are created to correct the residual errors in the predictions from the existing sequence of trees.The effect is that the model can quickly fit , then overfit the training data. The learning rate is a weight factor for controling the model to make less/more corrections for each tree added to the model. The right model uses 0.9 learning rate and 0.9 fraction, that makes the model quickly fit so it doesn’t need more predictors. The left model uses 0.1 learning rate and 0.9 fraction, that makes the model fitting more predictors.

<https://machinelearningmastery.com/tune-learning-rate-for-gradient-boosting-with-xgboost-in-python/>

1. Which model do you think would be more predictive of other samples?

The left model with 0.1 learning rate has the effect of making less corrections for each tree added to the model. This in turn results in more trees that must be added to the model. So the left model is more predictive of other samples.

1. How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24?

An increase in interaction depth (tree depth) will increase the number of predictors and it is likely to spread importance more evenly across predictors.

### 8.7

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

1. Which tree-based regression model gives the optimal resampling and test set performance?

We will use the same pre-processing steps and validation approach as in Exercises 6.3 and 7.5. The following tree-based models will be used: Single Decision Tree Model(CART), Random Forest, GBM, and Cubist models.

library(vip)  
library(RANN)  
library(AppliedPredictiveModeling)  
library(dplyr)  
library(tidyr)  
library(e1071)  
library(MASS)  
library(caret)  
library(corrplot)  
library(ggplot2)

data(ChemicalManufacturingProcess)  
pp <- preProcess(ChemicalManufacturingProcess, method = c("center","scale", "knnImpute", "nzv"))  
ChemicalManufacturingProcess<- predict(pp, ChemicalManufacturingProcess)

set.seed(100)  
training.samples <- ChemicalManufacturingProcess$Yield %>%  
createDataPartition(p = 0.8, list = FALSE)  
train.data <- ChemicalManufacturingProcess[training.samples, ]  
test.data <- ChemicalManufacturingProcess[-training.samples, ]  
# using same train control as for ex. 6.3 and 7.5.  
ctrl<-trainControl("cv", number = 10)

### Classification and Regression Tree (CART)

A decision tree is a largely used non-parametric (no underlying assumptions about the distribution of the errors or the data) effective machine learning modeling technique for regression and classification problems. To find solutions a decision tree makes sequential, hierarchical decision about the outcomes variable based on the predictor data.

The main parameter that is needed to be tuned is a max tree depth - the depth of a decision tree is the length of the longest path from a root to a leaf. We will find the optimal max tree depth using 10 folds CV.

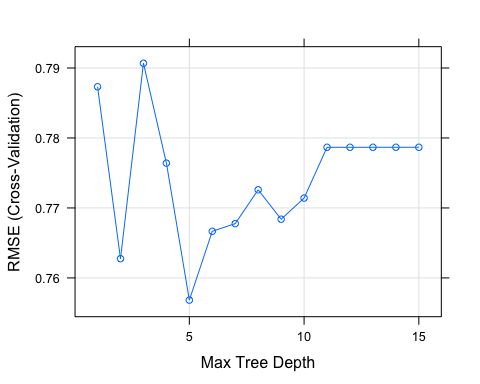
set.seed(100)  
cart\_model <- train(Yield~., data = train.data, method = "rpart2", trControl = ctrl, tuneGrid = expand.grid(maxdepth = 1:15))  
  
cart\_model

## CART   
##   
## 144 samples  
## 56 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...   
## Resampling results across tuning parameters:  
##   
## maxdepth RMSE Rsquared MAE   
## 1 0.7873160 0.3715603 0.6346939  
## 2 0.7627480 0.4147724 0.6140879  
## 3 0.7906780 0.3868215 0.6293701  
## 4 0.7764044 0.4001289 0.6173047  
## 5 0.7568251 0.4313976 0.6126941  
## 6 0.7666613 0.4301931 0.6202988  
## 7 0.7677603 0.4354624 0.6113529  
## 8 0.7725921 0.4219244 0.6142386  
## 9 0.7683856 0.4346920 0.6023534  
## 10 0.7714092 0.4359774 0.5993819  
## 11 0.7786709 0.4289863 0.6068793  
## 12 0.7786709 0.4289863 0.6068793  
## 13 0.7786709 0.4289863 0.6068793  
## 14 0.7786709 0.4289863 0.6068793  
## 15 0.7786709 0.4289863 0.6068793  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was maxdepth = 5.

cart\_model$bestTune

## maxdepth  
## 5 5

plot(cart\_model)



# making predictions on the test set  
predictions <- cart\_model %>% predict(test.data)  
# checking RMSE on the test set  
cart\_RMSE = caret::RMSE(predictions, test.data$Yield)  
cart\_RMSE

## [1] 0.9441681

The optimal max tree depth is 5. The optimal selected model has RMSE = 0.9441681 on the test set.

### Bagged Tree Model

Bagging is a simple ensembling technique in which we build many independent predictors/models/learners and combine them using some model averaging techniques.

There are no paramentrs to tune.

# treebag method does not have tuning parametrs, CV will not be used  
set.seed(100)  
bagged\_model <- train(Yield~., data = train.data, method = "treebag", trControl = trainControl(method = 'none'))  
  
# making predictions on the test set  
predictions <- bagged\_model %>% predict(test.data)  
# checking RMSE on the test set  
bagged\_RMSE = caret::RMSE(predictions, test.data$Yield)  
bagged\_RMSE

## [1] 0.8108503

### Random Forest

Random forest is an ensemble method which tries to build multiple CART models with different samples and different initial variables. It will repeat the process several times and then make a final prediction on each observation. This final prediction can simply be the mean of each prediction.

The main tuning parametr is mtry - number of variable is randomly collected to be sampled at each split time.

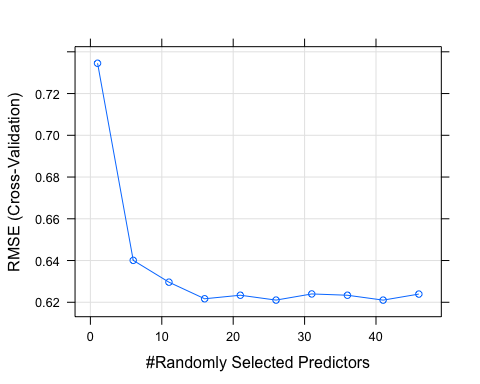
set.seed(100)  
rfGrid <- expand.grid(mtry=seq(1,50,by=5))  
rf\_model <- train(Yield~., data = train.data, method = "rf",  
 tuneGrid = rfGrid,  
 importance = TRUE,  
 trControl = ctrl, ntree = 200)  
  
rf\_model

## Random Forest   
##   
## 144 samples  
## 56 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...   
## Resampling results across tuning parameters:  
##   
## mtry RMSE Rsquared MAE   
## 1 0.7345123 0.5011027 0.5865249  
## 6 0.6400882 0.5980802 0.5070327  
## 11 0.6296269 0.6007108 0.4885491  
## 16 0.6216800 0.6056202 0.4872770  
## 21 0.6233745 0.5968384 0.4812310  
## 26 0.6210404 0.5971460 0.4781654  
## 31 0.6239887 0.5939719 0.4774513  
## 36 0.6233681 0.5905113 0.4795201  
## 41 0.6210270 0.5932933 0.4781704  
## 46 0.6239234 0.5873359 0.4777321  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was mtry = 41.

rf\_model$bestTune

## mtry  
## 9 41

plot(rf\_model)



# making predictions on the test set  
predictions <- rf\_model %>% predict(test.data)  
# checking RMSE on the test set  
rf\_RMSE = caret::RMSE(predictions, test.data$Yield)  
rf\_RMSE

## [1] 0.7390481

The optimal number of variable is randomly collected to be sampled at each split time is 41. Using this value model achived RMSE = 0.7390481 on the test set.

### Boosted Tree Model

Boosted Tree Model utilizes an ensemble technique in which the predictors are not made independently, but sequentially. This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Therefore, the observations have an unequal probability of appearing in subsequent models and ones with the highest error appear most. interaction.depth - number of splits it has to perform on a tree (starting from a single node).

n.trees - integer specifying the total number of trees to fit. This is equivalent to the number of iterations.

shrinkage - learning rate.

minobsinnode - integer specifying the minimum number of observations in the terminal nodes of the trees.

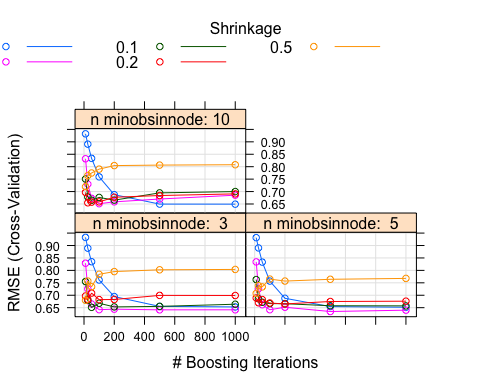
set.seed(100)  
gbmGrid <- expand.grid(interaction.depth=seq(1,5,10), n.trees=c(10, 25,50,100,200,500,1000), shrinkage=c(0.01,0.05,0.1,0.2, 0.5), n.minobsinnode = c(3,5,10))  
  
gb\_model <- train(Yield~., data = train.data, method = "gbm", tuneGrid = gbmGrid, trControl = ctrl, verbose = FALSE)  
gb\_model

## Stochastic Gradient Boosting   
##   
## 144 samples  
## 56 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...   
## Resampling results across tuning parameters:  
##   
## shrinkage n.minobsinnode n.trees RMSE Rsquared MAE   
## 0.01 3 10 0.9326100 0.4618374 0.7628944  
## 0.01 3 25 0.8891746 0.4972715 0.7256504  
## 0.01 3 50 0.8348727 0.5164236 0.6781761  
## 0.01 3 100 0.7612291 0.5349862 0.6164657  
## 0.01 3 200 0.6946697 0.5469765 0.5573006  
## 0.01 3 500 0.6551679 0.5581905 0.5137190  
## 0.01 3 1000 0.6530228 0.5567898 0.5125913  
## 0.01 5 10 0.9316456 0.4533477 0.7620302  
## 0.01 5 25 0.8902804 0.4844414 0.7257657  
## 0.01 5 50 0.8332489 0.5062260 0.6768553  
## 0.01 5 100 0.7569918 0.5383567 0.6134140  
## 0.01 5 200 0.6878006 0.5530089 0.5516602  
## 0.01 5 500 0.6543522 0.5566127 0.5123205  
## 0.01 5 1000 0.6516945 0.5593989 0.5078468  
## 0.01 10 10 0.9321672 0.4798041 0.7616266  
## 0.01 10 25 0.8906255 0.4980368 0.7261444  
## 0.01 10 50 0.8335371 0.5285587 0.6762679  
## 0.01 10 100 0.7591633 0.5420960 0.6137327  
## 0.01 10 200 0.6874620 0.5557744 0.5493453  
## 0.01 10 500 0.6495604 0.5605194 0.5115241  
## 0.01 10 1000 0.6492951 0.5586330 0.5098887  
## 0.05 3 10 0.8287788 0.4911482 0.6706672  
## 0.05 3 25 0.7258933 0.5724583 0.5849018  
## 0.05 3 50 0.6648330 0.5823110 0.5295694  
## 0.05 3 100 0.6419532 0.5826816 0.4999916  
## 0.05 3 200 0.6439615 0.5795469 0.5079445  
## 0.05 3 500 0.6412999 0.5840102 0.5061506  
## 0.05 3 1000 0.6414183 0.5882987 0.5062085  
## 0.05 5 10 0.8341370 0.4904356 0.6796123  
## 0.05 5 25 0.7258113 0.5638215 0.5869298  
## 0.05 5 50 0.6618014 0.5770819 0.5269245  
## 0.05 5 100 0.6417883 0.5804795 0.5013045  
## 0.05 5 200 0.6516743 0.5631999 0.5121836  
## 0.05 5 500 0.6345575 0.5868624 0.4936278  
## 0.05 5 1000 0.6399489 0.5872903 0.5012100  
## 0.05 10 10 0.8318558 0.5293105 0.6749510  
## 0.05 10 25 0.7297601 0.5598707 0.5912767  
## 0.05 10 50 0.6726900 0.5610097 0.5371086  
## 0.05 10 100 0.6512569 0.5629725 0.5069193  
## 0.05 10 200 0.6586188 0.5546886 0.5161339  
## 0.05 10 500 0.6700156 0.5457953 0.5302607  
## 0.05 10 1000 0.6854041 0.5374469 0.5484420  
## 0.10 3 10 0.7552913 0.5347628 0.6111461  
## 0.10 3 25 0.6791465 0.5476922 0.5407217  
## 0.10 3 50 0.6513554 0.5689464 0.5048736  
## 0.10 3 100 0.6677350 0.5336776 0.5149925  
## 0.10 3 200 0.6531531 0.5546486 0.5134025  
## 0.10 3 500 0.6552096 0.5618202 0.5196023  
## 0.10 3 1000 0.6639323 0.5585209 0.5195337  
## 0.10 5 10 0.7630633 0.5125377 0.6121506  
## 0.10 5 25 0.6861336 0.5319470 0.5458579  
## 0.10 5 50 0.6716457 0.5293014 0.5309055  
## 0.10 5 100 0.6664458 0.5363098 0.5264536  
## 0.10 5 200 0.6659733 0.5476132 0.5281060  
## 0.10 5 500 0.6577736 0.5637314 0.5204511  
## 0.10 5 1000 0.6572244 0.5709271 0.5132263  
## 0.10 10 10 0.7505063 0.5216711 0.6065167  
## 0.10 10 25 0.6786444 0.5347436 0.5292919  
## 0.10 10 50 0.6649873 0.5352772 0.5133201  
## 0.10 10 100 0.6763870 0.5247135 0.5281146  
## 0.10 10 200 0.6654704 0.5458569 0.5229492  
## 0.10 10 500 0.6947404 0.5278389 0.5513645  
## 0.10 10 1000 0.7000075 0.5301968 0.5510066  
## 0.20 3 10 0.6969294 0.5335299 0.5597385  
## 0.20 3 25 0.6831936 0.5151690 0.5411742  
## 0.20 3 50 0.7085832 0.4829993 0.5693178  
## 0.20 3 100 0.6824570 0.5193147 0.5379730  
## 0.20 3 200 0.6829268 0.5182025 0.5334239  
## 0.20 3 500 0.6994064 0.5053553 0.5442777  
## 0.20 3 1000 0.6990000 0.5089999 0.5415651  
## 0.20 5 10 0.6890591 0.5248098 0.5514406  
## 0.20 5 25 0.6698352 0.5332545 0.5367447  
## 0.20 5 50 0.6826614 0.5154795 0.5501433  
## 0.20 5 100 0.6687844 0.5458415 0.5346022  
## 0.20 5 200 0.6650106 0.5556668 0.5262343  
## 0.20 5 500 0.6748447 0.5492472 0.5401798  
## 0.20 5 1000 0.6761038 0.5476683 0.5367923  
## 0.20 10 10 0.6966758 0.5361067 0.5621196  
## 0.20 10 25 0.6548587 0.5594956 0.5165226  
## 0.20 10 50 0.6573538 0.5526807 0.5238301  
## 0.20 10 100 0.6584822 0.5532762 0.5173706  
## 0.20 10 200 0.6771787 0.5441255 0.5372116  
## 0.20 10 500 0.6840199 0.5399476 0.5408927  
## 0.20 10 1000 0.6907288 0.5323118 0.5450324  
## 0.50 3 10 0.6796575 0.5129535 0.5479244  
## 0.50 3 25 0.7570598 0.4346312 0.6090417  
## 0.50 3 50 0.7341522 0.4784759 0.6077299  
## 0.50 3 100 0.7842686 0.4561102 0.6273743  
## 0.50 3 200 0.7952141 0.4539325 0.6336392  
## 0.50 3 500 0.8023561 0.4504225 0.6449130  
## 0.50 3 1000 0.8038108 0.4498260 0.6463488  
## 0.50 5 10 0.7075398 0.4891099 0.5493320  
## 0.50 5 25 0.7384140 0.4809038 0.5907278  
## 0.50 5 50 0.7335918 0.4775632 0.5962349  
## 0.50 5 100 0.7649627 0.4434535 0.6086084  
## 0.50 5 200 0.7570283 0.4565072 0.5992297  
## 0.50 5 500 0.7641690 0.4566922 0.5947079  
## 0.50 5 1000 0.7677601 0.4526703 0.5982430  
## 0.50 10 10 0.7197366 0.4843065 0.5781936  
## 0.50 10 25 0.7642876 0.4570747 0.6233435  
## 0.50 10 50 0.7746315 0.4574106 0.6410178  
## 0.50 10 100 0.7902085 0.4516446 0.6477950  
## 0.50 10 200 0.8044419 0.4554960 0.6534176  
## 0.50 10 500 0.8064442 0.4513343 0.6426350  
## 0.50 10 1000 0.8078960 0.4496734 0.6415419  
##   
## Tuning parameter 'interaction.depth' was held constant at a value of 1  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were n.trees = 500,  
## interaction.depth = 1, shrinkage = 0.05 and n.minobsinnode = 5.

gb\_model$bestTune

## n.trees interaction.depth shrinkage n.minobsinnode  
## 34 500 1 0.05 5

plot(gb\_model)



# making predictions on the test set  
predictions <- gb\_model %>% predict(test.data)  
# checking RMSE on the test set  
gb\_RMSE = caret::RMSE(predictions, test.data$Yield)  
gb\_RMSE

## [1] 0.6769613

The optimal tuning parametrs identified via 10 folds CV are the following:

n.trees = 500

interaction.depth = 1

shrinkage = 0.05

n.minobsinnode = 5

Tuned Boosted Tree Model has RMSE = 0.6769613 on the test set.

### Rule-Based Cubist

Cubist is a rule–based model that is an extension of Quinlan’s M5 model tree. A tree is grown where the terminal leaves contain linear regression models. These models are based on the predictors used in previous splits. Also, there are intermediate linear models at each step of the tree. A prediction is made using the linear regression model at the terminal node of the tree, but is “smoothed” by taking into account the prediction from the linear model in the previous node of the tree (which also occurs recursively up the tree). The tree is reduced to a set of rules, which initially are paths from the top of the tree to the bottom. Rules are eliminated via pruning and/or combined for simplification.

Tuning parameters:

committees define how many committee models (e.g.. boosting iterations) should be used.

neighbors - number of nearest neighbors. Using nearest–neighbors to adjust the predictions from the rule–based model. First, a model tree (with or without committees) is created. Once a sample is predicted by this model, Cubist can find it’s nearest neighbors and determine the average of these training set points.

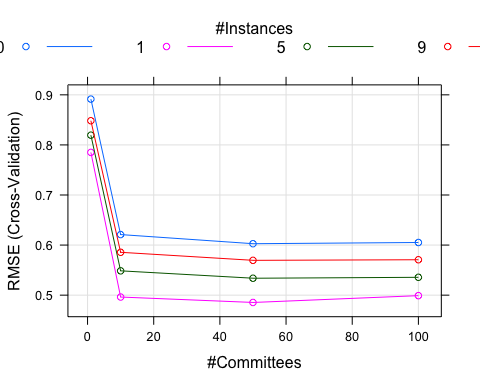
set.seed(100)  
cubistGrid <- expand.grid(committees = c(1, 10, 50, 100), neighbors = c(0, 1, 5, 9))  
cubist\_model <- train( Yield~., data = train.data, method = "cubist", verbose = FALSE, tuneGrid = cubistGrid, trControl = ctrl)  
  
cubist\_model

## Cubist   
##   
## 144 samples  
## 56 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...   
## Resampling results across tuning parameters:  
##   
## committees neighbors RMSE Rsquared MAE   
## 1 0 0.8914553 0.4461661 0.6435026  
## 1 1 0.7850817 0.6020537 0.5004690  
## 1 5 0.8194323 0.5471521 0.5687479  
## 1 9 0.8483137 0.5026285 0.5970908  
## 10 0 0.6209711 0.5849675 0.4983110  
## 10 1 0.4962381 0.7406846 0.3701181  
## 10 5 0.5485671 0.6819350 0.4436434  
## 10 9 0.5855791 0.6346287 0.4783854  
## 50 0 0.6027289 0.6150887 0.4774839  
## 50 1 0.4853885 0.7489295 0.3604592  
## 50 5 0.5337756 0.7026580 0.4389415  
## 50 9 0.5695131 0.6595521 0.4679022  
## 100 0 0.6051706 0.6140775 0.4844934  
## 100 1 0.4990085 0.7471510 0.3699747  
## 100 5 0.5356701 0.7051548 0.4406408  
## 100 9 0.5707245 0.6611166 0.4712630  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were committees = 50 and neighbors = 1.

cubist\_model$bestTune

## committees neighbors  
## 10 50 1

plot(cubist\_model)



# making predictions on the test set  
predictions <- cubist\_model %>% predict(test.data)  
# checking RMSE on the test set  
cubist\_RMSE = caret::RMSE(predictions, test.data$Yield)  
cubist\_RMSE

## [1] 0.5082907

The following optimal paramters were selected via 10 folds CV: committees - 50 and neighbors - 1

Models’ performance summary

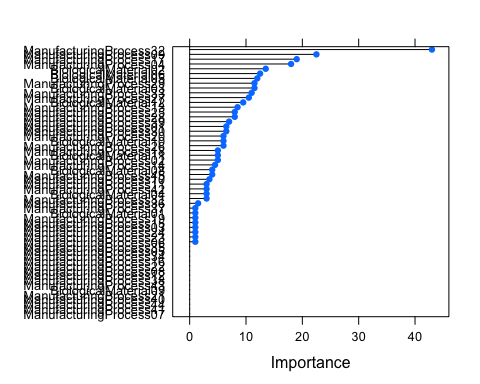
m1<-cbind(RMSE=cart\_RMSE)  
m2<-cbind(RMSE=bagged\_RMSE)  
m3<-cbind(RMSE=rf\_RMSE)  
m4<-cbind(RMSE=gb\_RMSE)  
m5<-cbind(RMSE=cubist\_RMSE)  
summary = rbind(m1, m2, m3, m4, m5)  
rownames(summary) <- c("CART", 'Bagged',"RF","Boosted","Cubist")  
summary

## RMSE  
## CART 0.9441681  
## Bagged 0.8108503  
## RF 0.7390481  
## Boosted 0.6769613  
## Cubist 0.5082907

The cubist model performed the best having the lowest RMSE on the test set. Also the cubist model provides the best performance among all models we have tuned in Exercises 6.3 and 7.5.

1. 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 nonlinear models?

importance <- varImp(cubist\_model, scale=FALSE)  
plot(importance)



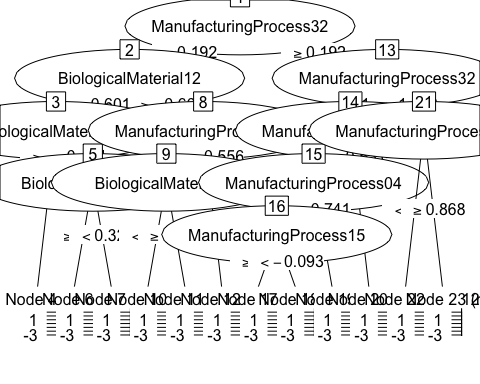
importance

## cubist variable importance  
##   
## only 20 most important variables shown (out of 56)  
##   
## Overall  
## ManufacturingProcess32 43.0  
## ManufacturingProcess09 22.5  
## ManufacturingProcess17 19.0  
## ManufacturingProcess04 18.0  
## BiologicalMaterial02 13.5  
## BiologicalMaterial06 12.5  
## BiologicalMaterial05 12.0  
## BiologicalMaterial03 11.5  
## ManufacturingProcess29 11.5  
## ManufacturingProcess33 11.0  
## ManufacturingProcess27 10.5  
## BiologicalMaterial12 9.5  
## ManufacturingProcess13 8.5  
## ManufacturingProcess25 8.0  
## ManufacturingProcess28 8.0  
## ManufacturingProcess39 7.0  
## ManufacturingProcess30 6.5  
## ManufacturingProcess01 6.5  
## BiologicalMaterial10 6.0  
## ManufacturingProcess20 6.0

ManufacturingProcess32, ManufacturingProcess09, ManufacturingProcess17 and ManufacturingProcess04 are the most important predictors. ManufacturingProcess dominates over Biological in the results of our Cubist model as well as in other models tested before (in ex. 6.3 and 7.5.). The top 10 most important predictors of Cubist model are similar to the previous models results.

1. 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?

plot(partykit::as.party(cart\_model$finalModel))



The visualization of the Single Decision Tree Model shows that Manufacturing process 32 is the most important variable and that dominating process is manufacturing.