Part 2 Homework

Packages used

library(AppliedPredictiveModeling)  
library(Amelia)  
library(doParallel)  
library(missForest)  
library(cwhmisc)  
library(MASS)  
library(forecast)  
library(tidyverse)  
library(pls)  
library(caret)  
library(data.table)  
library(psych)  
library(mlbench)  
library(nnet)  
library(earth)  
library(kernlab)  
library(Metrics)

# Chapter 6 (HW 6.3)

A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is the understand the relationship between biological measurements of the raw materials (predictors), measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1 % will boost revenue by approximately one hundred thousand dollars per batch.

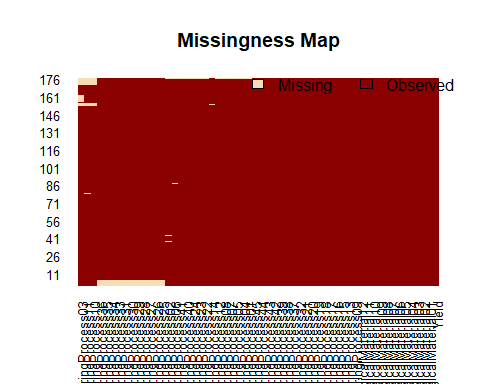
1. Start $ and use the commands to load the data:

library(AppliedPredictiveModeling)  
data("ChemicalManufacturingProcess")

The matrix processPredictors contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. yield contains the percent yield for each run.

1. A small percentage of cells in the predictor set contains missing values. Use an imputation function to fill in these missing values.

library(Amelia)  
missmap(ChemicalManufacturingProcess)



It appears that the missing values are quite minor and seem to be missing at random.

library(doParallel)  
library(missForest)  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100, parallelize = "forests")$ximp

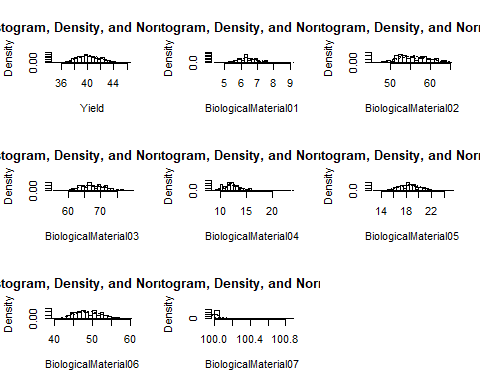
## missForest iteration 1 in progress...done!  
## missForest iteration 2 in progress...done!  
## missForest iteration 3 in progress...done!

data.set <- list(org.data.set = chem.imput)

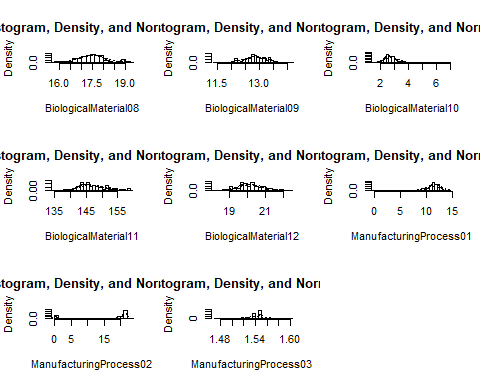
1. Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter. What is the optimal value of the performance metric?

Lets explore the distribution of the variables

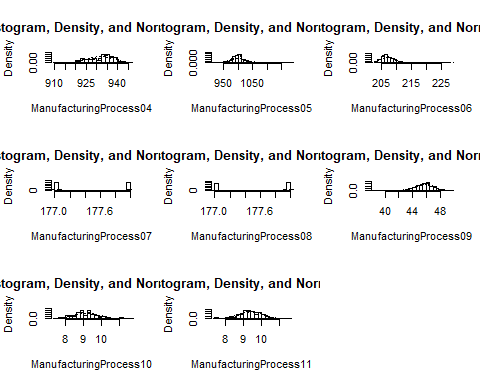
library(psych)  
  
multi.hist(data.set$org.data.set[1:8])



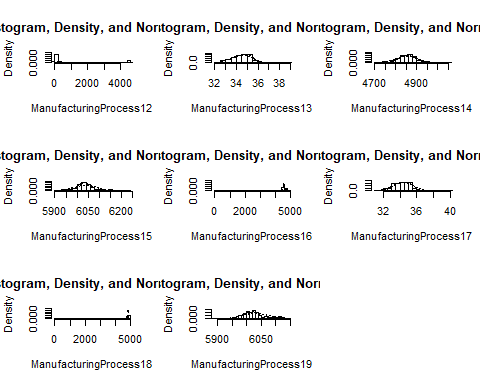
multi.hist(data.set$org.data.set[9:16])



multi.hist(data.set$org.data.set[17:24])



multi.hist(data.set$org.data.set[25:32])



The distributions of several of the variables are noticeably skewed, so we will obtain the box cox transformation lambda

variables.to.transform <- c("BiologicalMaterial04", "ManufacturingProcess01",  
 "ManufacturingProcess02","ManufacturingProcess03",  
 "ManufacturingProcess06", "ManufacturingProcess07",  
 "ManufacturingProcess08","ManufacturingProcess12",  
 "ManufacturingProcess16","ManufacturingProcess18")   
  
results <- NULL  
for (i in 1:length(variables.to.transform)){  
 results[[i]] <- list(variable = variables.to.transform[[i]],  
 lambda = BoxCox.lambda(data.set$org.data.set[,variables.to.transform[[i]]]))}  
results

## [[1]]  
## [[1]]$variable  
## [1] "BiologicalMaterial04"  
##   
## [[1]]$lambda  
## [1] -0.9999242  
##   
##   
## [[2]]  
## [[2]]$variable  
## [1] "ManufacturingProcess01"  
##   
## [[2]]$lambda  
## [1] 1.999959  
##   
##   
## [[3]]  
## [[3]]$variable  
## [1] "ManufacturingProcess02"  
##   
## [[3]]$lambda  
## [1] 1.999959  
##   
##   
## [[4]]  
## [[4]]$variable  
## [1] "ManufacturingProcess03"  
##   
## [[4]]$lambda  
## [1] -0.9999242  
##   
##   
## [[5]]  
## [[5]]$variable  
## [1] "ManufacturingProcess06"  
##   
## [[5]]$lambda  
## [1] -0.9999242  
##   
##   
## [[6]]  
## [[6]]$variable  
## [1] "ManufacturingProcess07"  
##   
## [[6]]$lambda  
## [1] -0.9999242  
##   
##   
## [[7]]  
## [[7]]$variable  
## [1] "ManufacturingProcess08"  
##   
## [[7]]$lambda  
## [1] 1.999924  
##   
##   
## [[8]]  
## [[8]]$variable  
## [1] "ManufacturingProcess12"  
##   
## [[8]]$lambda  
## [1] 4.102259e-05  
##   
##   
## [[9]]  
## [[9]]$variable  
## [1] "ManufacturingProcess16"  
##   
## [[9]]$lambda  
## [1] 1.999959  
##   
##   
## [[10]]  
## [[10]]$variable  
## [1] "ManufacturingProcess18"  
##   
## [[10]]$lambda  
## [1] 1.999959

We have our lambda transformations and we can apply them to our data set then set our test and train splices.

trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))

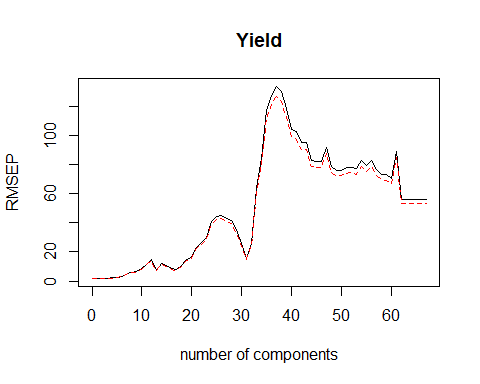
Splitting the test and training data set

set.seed(123)

data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))

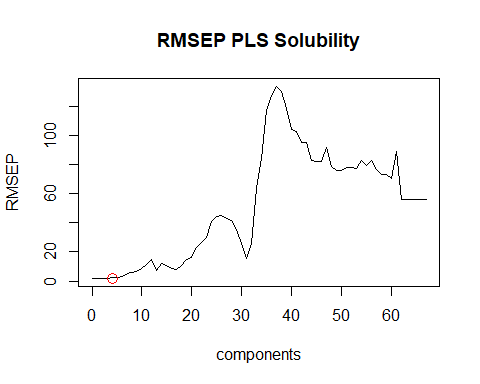
Now we can proceed with the model of choice which is the Partial Least Squares. Note that the evaluation below relied heavily on this rpubs post- <https://rpubs.com/omicsdata/pls>.

library(pls)  
chem.plsFit <- plsr(Yield ~ ., data = data.set$train, validation = "CV")  
validationplot(chem.plsFit, val.type="RMSEP")



We see a steep increase at 10 components followed by many mountains and valleys. Our intention is to select the most optimal number of components to minimize the RMSE.

pls.RMSEP <- RMSEP(chem.plsFit, estimate="CV")   
plot(pls.RMSEP, main="RMSEP PLS Solubility", xlab="components")  
min\_comp <- which.min(pls.RMSEP$val)  
points(min\_comp, min(pls.RMSEP$val), pch=1, col="red", cex=1.5)



Our optimal number of components is 4.

[1] RPubs - Partial Least Square Regression. N.p., n.d. Web. 14 Nov. 2017. - <https://rpubs.com/omicsdata/pls>

1. Predict the response for the test set. What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

Our results from our training set

chem.plsPredict.train <- predict(chem.plsFit, data = data.set$train, ncomp = min\_comp)  
pls.eval <- data.frame(obs = data.set$train[,1], pred = chem.plsPredict.train[,1,1])  
defaultSummary(pls.eval)

## RMSE Rsquared MAE   
## 1.7102647 0.1361341 1.3523163

Results from the test set

chem.plsPredict <- predict(chem.plsFit, data = data.set$test, ncomp = min\_comp)  
pls.eval <- data.frame(obs = data.set$test[,1], pred = chem.plsPredict[,1,1])  
defaultSummary(pls.eval)

## RMSE Rsquared MAE   
## 1.918598471 0.004722081 1.590453088

While our RMSE is within range of both the training and test data set the Rsquared is significantly less for the test set suggesting a model that does not explain much of the variation. This may not be the most optimal model.

1. Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

library(data.table)  
dt <- setDT(varImp(chem.plsFit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: BiologicalMaterial04.boxcoxtrans 32.368582601000  
## 2: ManufacturingProcess03.boxcoxtrans 8.506741877897  
## 3: ManufacturingProcess36 5.806260614702  
## 4: ManufacturingProcess40 3.266626682782  
## 5: ManufacturingProcess41 2.639361307649  
## 6: ManufacturingProcess03 2.269895183599  
## 7: ManufacturingProcess29 1.352259516569  
## 8: ManufacturingProcess45 1.201727828851  
## 9: ManufacturingProcess34 0.794194810633  
## 10: BiologicalMaterial08 0.752963253106  
## 11: BiologicalMaterial12 0.668004792331  
## 12: BiologicalMaterial04 0.546936039550  
## 13: BiologicalMaterial10 0.523925218873  
## 14: ManufacturingProcess21 0.398347227817  
## 15: ManufacturingProcess30 0.386390369135  
## 16: ManufacturingProcess11 0.371122495753  
## 17: BiologicalMaterial01 0.367740795619  
## 18: BiologicalMaterial07 0.330266386466  
## 19: ManufacturingProcess17 0.329128566762  
## 20: ManufacturingProcess43 0.327192135031  
## 21: ManufacturingProcess02 0.298854267291  
## 22: ManufacturingProcess37 0.231417585786  
## 23: BiologicalMaterial06 0.214749503138  
## 24: ManufacturingProcess39 0.214642665238  
## 25: ManufacturingProcess44 0.191312023749  
## 26: ManufacturingProcess09 0.182069206380  
## 27: ManufacturingProcess10 0.172273127562  
## 28: ManufacturingProcess13 0.148855072823  
## 29: ManufacturingProcess33 0.147106292705  
## 30: ManufacturingProcess38 0.134231180788  
## 31: ManufacturingProcess01 0.126966997906  
## 32: BiologicalMaterial05 0.113190008984  
## 33: ManufacturingProcess31 0.101924484749  
## 34: BiologicalMaterial02 0.093109130289  
## 35: ManufacturingProcess32 0.080188691046  
## 36: ManufacturingProcess42 0.078552888846  
## 37: ManufacturingProcess23 0.068068009342  
## 38: ManufacturingProcess22 0.056054717778  
## 39: BiologicalMaterial03 0.051713043978  
## 40: BiologicalMaterial11 0.045882221016  
## 41: ManufacturingProcess02.boxcoxtrans 0.023989656138  
## 42: ManufacturingProcess28 0.022857404421  
## 43: ManufacturingProcess06 0.021422248447  
## 44: ManufacturingProcess24 0.019611881418  
## 45: ManufacturingProcess27 0.017232889324  
## 46: ManufacturingProcess18 0.016452028158  
## 47: ManufacturingProcess25 0.012718781692  
## 48: ManufacturingProcess35 0.012290521222  
## 49: ManufacturingProcess04 0.011998068738  
## 50: ManufacturingProcess19 0.011550833558  
## 51: ManufacturingProcess26 0.009768108092  
## 52: ManufacturingProcess01.boxcoxtrans 0.009352676122  
## 53: ManufacturingProcess20 0.007726498058  
## 54: ManufacturingProcess14 0.005445900561  
## 55: ManufacturingProcess16 0.003392532945  
## 56: ManufacturingProcess06.boxcoxtrans 0.003317659056  
## 57: ManufacturingProcess05 0.002734566621  
## 58: ManufacturingProcess15 0.002477274821  
## 59: ManufacturingProcess08.boxcoxtrans 0.000360698451  
## 60: ManufacturingProcess12.boxcoxtrans 0.000006545034  
## 61: ManufacturingProcess18.boxcoxtrans 0.000004004795  
## 62: ManufacturingProcess08 0.000002032898  
## 63: ManufacturingProcess16.boxcoxtrans 0.000001522387  
## 64: ManufacturingProcess07.boxcoxtrans -0.000005201508  
## 65: ManufacturingProcess12 -0.000016341601  
## 66: BiologicalMaterial09 -0.033911162468  
## 67: ManufacturingProcess07 -0.163814431627  
## rn Overall

We see that the Manufacturing Processes dominate the list for variable importance with a few biological materials having importance.

1. Explore the relationships between each of the top predictors and the response. How would this information be helpful in improving yield in future runs of the manufacturing process?

The top manufacturing process is ManufacturingProcess40

data.set$org.data.set %>% dplyr::select(Yield, ManufacturingProcess40) %>% cor() %>% .[2]

## [1] -0.04647267

We see a negative correlation ManufacturingProcess40 and Yield, which suggests that ManufacturingProcess40 represents a process that can decrease the Yield. We cannot say that ManufacturingProcess40 itself decrease Yield, only that a negative relationship exists.

The top biological process is BiologicalMaterial12

data.set$org.data.set %>% dplyr::select(Yield, BiologicalMaterial12) %>% cor() %>% .[2]

## [1] 0.3674976

We actually see the opposite for the Biological indicator in that there is a positive correlation to Yield. We cannot say that BiologicalMaterial12 itself improves Yield, only that a positive relationship exists.

# Chapter 7 Homework HW 7.2 and 7.5

## 7.2 Friedman (1991) introduced several benchmark data sets created by simulation. One of these simulations used the following nonlinear equation to create data:

where the ex values are random variables uniformly distributed between [0,1] (there are also 5 0other non-informative variables also created in simulation). The package mlbench contains a function called mlbench.friedman1 that simulates these data:

set.seed(1234)  
trainingData <- mlbench.friedman1(200, sd = 1)  
trainingData$x <- data.frame(trainingData$x)  
#featurePlot(trainingData$x, trainingData$y)  
testData <- mlbench.friedman1(5000, sd = 1)  
testData$x <- data.frame(testData$x)

Tune several models on these data.

Training Neural Net

nnetFit <- nnet(testData$x, testData$y,  
 size = 5,  
 decay = .01,  
 maxit = 500,  
 MaxNWts = 5 \* (ncol(testData$x) + 1) + 5 + 1)

## # weights: 61  
## initial value 1077271.618912   
## iter 10 value 1035722.717788  
## iter 20 value 1035626.108753  
## final value 1035625.899884   
## converged

results <- predict(nnetFit, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 14.36606

Training MARS

marsFit <- earth(testData$x, testData$y)  
results <- predict(marsFit, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 1.785523

Train K-Nearest Neighbors

This method at times will work and other times will not due to a known issue with the latest caret release. The error is “unable to find variable”optimismBoot“”, I am leaving the code in to show my attempt but note that this will not run appropriately.

#At one point I got the error optimismBoot not found  
knnTune <- train(testData$x, testData$y,   
 method = "knn",   
 preProc = c("center", "scale"))  
results <- predict(knnTune, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

Which models appear to give the best performance? Does MARS select informative predictors (those named X1-X5).

The model with the lowest performance metric that we are using is the RMSE value of 1.7855 for the MARS method.

varImp(marsFit)

## Overall  
## X4 100.00000  
## X1 78.68733  
## X2 63.36434  
## X5 44.56206  
## X3 31.99334  
## X6 0.00000  
## X7 0.00000  
## X8 0.00000  
## X9 0.00000  
## X10 0.00000

The MARS method did select variables with the most important of X1-X5 as shown in our previous table.

7.5 Exercise 6.3 describes data for a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several nonlinear regression models.

The below is the minimum steps needed to recreate the data set from Chapter 6. Please see problem from Chapter 6 for complete breakdown for each step and reasons behind the choices.

library(AppliedPredictiveModeling)  
data("ChemicalManufacturingProcess")

library(missForest)  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100)$ximp

## missForest iteration 1 in progress...done!  
## missForest iteration 2 in progress...done!  
## missForest iteration 3 in progress...done!  
## missForest iteration 4 in progress...done!

data.set <- list(org.data.set = chem.imput)

trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))

data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))

## Model Selection

Training Neural Net

nnetFit <- nnet(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 size = 5,  
 decay = .01,  
 maxit = 500)

## # weights: 346  
## initial value 211030.072431   
## iter 10 value 204166.716983  
## final value 204150.218586   
## converged

results <- predict(nnetFit,data.set$test %>% dplyr::select(-Yield)) %>% cbind(.,data.set$test$Yield) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 38.89715

Training MARS

marsFit <- earth(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield)  
results <- predict(marsFit, data.set$test %>% dplyr::select(-Yield)) %>% cbind(.,data.set$test$Yield) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 1.401761

Attempts to model a K-nearest neighbor was unsuccesful, the caret package appears to have a conflict with optimismboot and is unable to accomodate this data set. <https://stackoverflow.com/questions/46244763/caret-train-function-unable-to-find-variable-optimismboot>

1. Which nonlinear regression model gives the optimal re-sampling and test set performance?

The nonlinear regression model with the best test set performance was again the MARS model with an RMSE of 1.401761.

1. Which predictors are the most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

library(data.table)  
dt <- setDT(varImp(marsFit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

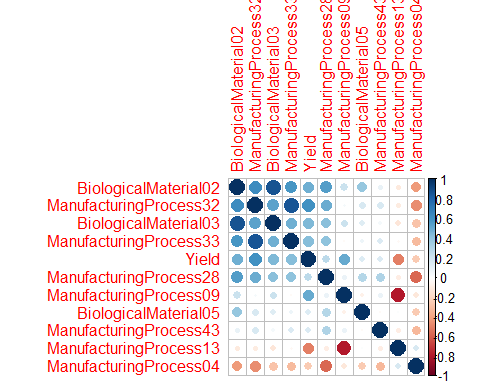
## rn Overall  
## 1: ManufacturingProcess32 100.00000  
## 2: ManufacturingProcess09 67.72174  
## 3: ManufacturingProcess13 42.33973  
## 4: ManufacturingProcess33 37.43641  
## 5: BiologicalMaterial02 37.25570  
## 6: BiologicalMaterial03 37.25570  
## 7: ManufacturingProcess28 31.32309  
## 8: ManufacturingProcess43 27.29428  
## 9: ManufacturingProcess04 19.81703  
## 10: BiologicalMaterial05 12.51240  
## 11: BiologicalMaterial01 0.00000  
## 12: BiologicalMaterial04 0.00000  
## 13: BiologicalMaterial06 0.00000  
## 14: BiologicalMaterial07 0.00000  
## 15: BiologicalMaterial08 0.00000  
## 16: BiologicalMaterial09 0.00000  
## 17: BiologicalMaterial10 0.00000  
## 18: BiologicalMaterial11 0.00000  
## 19: BiologicalMaterial12 0.00000  
## 20: ManufacturingProcess01 0.00000  
## 21: ManufacturingProcess02 0.00000  
## 22: ManufacturingProcess03 0.00000  
## 23: ManufacturingProcess05 0.00000  
## 24: ManufacturingProcess06 0.00000  
## 25: ManufacturingProcess07 0.00000  
## 26: ManufacturingProcess08 0.00000  
## 27: ManufacturingProcess10 0.00000  
## 28: ManufacturingProcess11 0.00000  
## 29: ManufacturingProcess12 0.00000  
## 30: ManufacturingProcess14 0.00000  
## 31: ManufacturingProcess15 0.00000  
## 32: ManufacturingProcess16 0.00000  
## 33: ManufacturingProcess17 0.00000  
## 34: ManufacturingProcess18 0.00000  
## 35: ManufacturingProcess19 0.00000  
## 36: ManufacturingProcess20 0.00000  
## 37: ManufacturingProcess21 0.00000  
## 38: ManufacturingProcess22 0.00000  
## 39: ManufacturingProcess23 0.00000  
## 40: ManufacturingProcess24 0.00000  
## 41: ManufacturingProcess25 0.00000  
## 42: ManufacturingProcess26 0.00000  
## 43: ManufacturingProcess27 0.00000  
## 44: ManufacturingProcess29 0.00000  
## 45: ManufacturingProcess30 0.00000  
## 46: ManufacturingProcess31 0.00000  
## 47: ManufacturingProcess34 0.00000  
## 48: ManufacturingProcess35 0.00000  
## 49: ManufacturingProcess36 0.00000  
## 50: ManufacturingProcess37 0.00000  
## 51: ManufacturingProcess38 0.00000  
## 52: ManufacturingProcess39 0.00000  
## 53: ManufacturingProcess40 0.00000  
## 54: ManufacturingProcess41 0.00000  
## 55: ManufacturingProcess42 0.00000  
## 56: ManufacturingProcess44 0.00000  
## 57: ManufacturingProcess45 0.00000  
## 58: BiologicalMaterial04.boxcoxtrans 0.00000  
## 59: ManufacturingProcess01.boxcoxtrans 0.00000  
## 60: ManufacturingProcess02.boxcoxtrans 0.00000  
## 61: ManufacturingProcess03.boxcoxtrans 0.00000  
## 62: ManufacturingProcess06.boxcoxtrans 0.00000  
## 63: ManufacturingProcess07.boxcoxtrans 0.00000  
## 64: ManufacturingProcess08.boxcoxtrans 0.00000  
## 65: ManufacturingProcess12.boxcoxtrans 0.00000  
## 66: ManufacturingProcess16.boxcoxtrans 0.00000  
## 67: ManufacturingProcess18.boxcoxtrans 0.00000  
## rn Overall

We see that the Manufacturing Processes dominate the list for variable importance with a few biological materials having importance. This is similar to our previous model however we do see a few more biological materials variables. Although we see none of the variables are previous model found important which suggests that the manufacturing processes contain a lot of information on the variance of yield.

1. Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

subset.data.set <- data.set$trans.data.set %>%   
 dplyr::select(ManufacturingProcess32, ManufacturingProcess09,   
 ManufacturingProcess13, ManufacturingProcess33,   
 BiologicalMaterial02, BiologicalMaterial03,   
 ManufacturingProcess28, ManufacturingProcess43,   
 ManufacturingProcess04, BiologicalMaterial05, Yield)

library(corrplot)  
correlations <- cor(subset.data.set)  
corrplot(correlations, order = "FPC", diag = TRUE)



The correlation plot provides some insight to these important variables. Yield has a mostly positive correlation with many of hte processes but has the strongest negative correlation with only manufacturing processes from our top 10 variables. We also see some very high correlations with other variables in our top variable set which suggests possible issues of multicollinearity.

library(tidyverse)

## 8.1

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"

1. Fit a random forest model to all of the predictors, then estimate the variable importance

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

## Overall  
## V1 8.66496174  
## V2 6.36504194  
## V3 0.76724347  
## V4 7.93742254  
## V5 2.10190958  
## V6 0.19154480  
## V7 0.04177869  
## V8 -0.08293296  
## V9 -0.06267951  
## V10 -0.04521494

Does the random forest model significantly use the uninformative predictors (V6-V10).

The random forest does not use the uniformative predictors. In fact, the variable importance is in line with the variable numbering.

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

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

## [1] 0.9386619

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?

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

library(data.table)  
dt <- setDT(varImp(model1, scale = FALSE), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: V4 6.59026253  
## 2: V2 6.04782043  
## 3: V1 5.34138782  
## 4: duplicate1 4.71267815  
## 5: V5 1.86532068  
## 6: V3 0.54776459  
## 7: V6 0.13680850  
## 8: V9 0.06233358  
## 9: V10 0.05506748  
## 10: V7 0.04745508  
## 11: V8 -0.01270436

We do see that the importance score has been reduced with the duplicate variable. The interaction between the two variables is reducing the signal we receive from using simple one of the variables.

simulated$duplicate2 <- simulated$V1 + rnorm(200) \* .15  
cor(simulated$duplicate2, simulated$V1)

## [1] 0.8849327

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

library(data.table)  
dt <- setDT(varImp(model2, scale = FALSE), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: V4 7.292570850  
## 2: V2 6.562165756  
## 3: V1 4.562573722  
## 4: duplicate1 4.295437598  
## 5: V5 2.070957251  
## 6: duplicate2 1.384370802  
## 7: V3 0.517422886  
## 8: V6 0.132806443  
## 9: V7 0.072280073  
## 10: V9 -0.003797135  
## 11: V10 -0.049610851  
## 12: V8 -0.082135816

We don’t any improvements are reductions in the variable importance but it is reducing variable importance of other variables that may be meaningful for other reasons like literature based reasons.

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 importance show the same pattern as the traditional random forest model?

library(party)  
fit.cforest <- cforest(y ~., data = simulated)

library(data.table)  
dt <- setDT(party::varimp(fit.cforest, conditional = TRUE) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-.)]

## rn .  
## 1: V4 5.281196888  
## 2: V2 4.373678395  
## 3: duplicate1 1.719885895  
## 4: V1 1.288320322  
## 5: V5 0.971063628  
## 6: duplicate2 0.276946982  
## 7: V3 0.021087453  
## 8: V6 0.019535747  
## 9: V7 0.011797289  
## 10: V10 0.002561214  
## 11: V9 -0.001990153  
## 12: V8 -0.005674798

dt <- setDT(party::varimp(fit.cforest, conditional = FALSE) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-.)]

## rn .  
## 1: V4 6.57619578  
## 2: V2 5.79417540  
## 3: duplicate1 4.99198355  
## 4: V1 3.74038496  
## 5: V5 1.62301336  
## 6: duplicate2 1.02429697  
## 7: V7 0.04711682  
## 8: V9 0.02985211  
## 9: V6 0.01570609  
## 10: V3 0.01140049  
## 11: V10 -0.01415559  
## 12: V8 -0.03169090

The top few variables remain at the same level of importance for both models. However, the variable importance is less using conditional inference trees via cpart from the party package.

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

library(Cubist)  
fit.cubist <- cubist(simulated %>% select(-y), simulated$y, committees = 4)  
fit.cubist

##   
## Call:  
## cubist.default(x = simulated %>% select(-y), y = simulated$y, committees  
## = 4)  
##   
## Number of samples: 200   
## Number of predictors: 12   
##   
## Number of committees: 4   
## Number of rules per committee: 1, 4, 2, 5

dt <- setDT(varImp(fit.cubist) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: V2 64.0  
## 2: V1 63.5  
## 3: V4 50.0  
## 4: V5 49.0  
## 5: V3 35.0  
## 6: duplicate1 25.5  
## 7: duplicate2 7.5  
## 8: V6 4.0  
## 9: V7 0.0  
## 10: V8 0.0  
## 11: V9 0.0  
## 12: V10 0.0

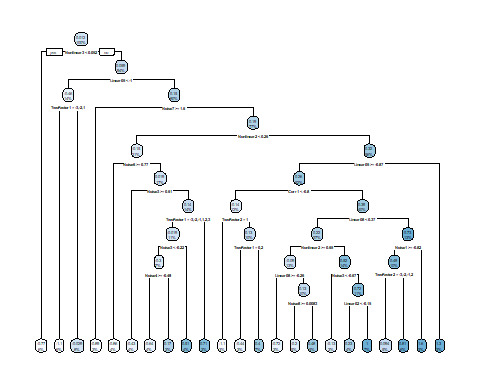
Using the cubist model we actually get very different results. In fact, the model appears to adjust for the highly correlated variables and gives us variable importance that more closely match our original data set before the added variables.

## 8.2

Use a simulation to show tree bias with different granularity.

library(mlbench)  
library(tidyverse)  
library(rpart)  
library(rpart.plot)  
library(caret)  
  
data.set <- twoClassSim(300, noiseVars = 8, corrVar = 6, corrValue = 0.8) %>%  
 mutate(TwoFactor1 = as.factor(round(TwoFactor1, 0)),  
 TwoFactor2 = as.factor(round(TwoFactor2, 0)))

r.fit <- rpart(Linear01 ~ ., data=data.set)  
rpart.plot(r.fit)



library(data.table)  
dt <- setDT(varImp(r.fit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: TwoFactor2 1.62598157  
## 2: TwoFactor1 0.92692954  
## 3: Noise1 0.76886577  
## 4: Linear09 0.72470303  
## 5: Linear08 0.71648162  
## 6: Noise5 0.69175952  
## 7: Noise6 0.61321134  
## 8: Noise3 0.59730960  
## 9: Linear02 0.59195971  
## 10: Noise4 0.54439475  
## 11: Linear05 0.48916749  
## 12: Linear07 0.47884757  
## 13: Corr5 0.46520871  
## 14: Linear03 0.43772882  
## 15: Linear10 0.43364590  
## 16: Linear04 0.43274643  
## 17: Nonlinear2 0.38760350  
## 18: Nonlinear3 0.38327928  
## 19: Corr2 0.36396153  
## 20: Noise8 0.31716985  
## 21: Corr3 0.31250548  
## 22: Corr6 0.28603545  
## 23: Noise7 0.24357693  
## 24: Corr1 0.19694726  
## 25: Linear06 0.17564253  
## 26: Nonlinear1 0.15408339  
## 27: Noise2 0.08328166  
## 28: Corr4 0.00000000  
## 29: Class 0.00000000  
## rn Overall

We can see the factor variables are not as important as the linear variables. Trees will typically bias towards linear variables as they give more opportunties to split a tree as opposed to factor/categorical variables. However, it’s important to know if the splits make sense for the data domain or not.

## 8.3

In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by 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-had 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?

A learning rate closer to 1 will make less corrections for each tree added to the model. A high bagging fraction closer to 1 indicates more of the data is used which can lead to overfitting. So as the learning rate and bagging fraction increase, the importance will be placed on fewer and fewer predictors.

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

Subsequently, if the parameters increase the model performance will then decrease. Therefore, the model on the left will perform better. Although, I would be cautious about usefulness. If a model uses variables that can’t always be reliably captured than a model dependent on very few variables may be more powerful.

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

Interaction depth specifies the tree depth and node splits. As the tree depth increase, and more node splits occur the variable importance becomes spread across more predictors. In both models the variable importance would decrease for the top variables and increase for less important variables. If we have any highly correlated variables we may actually see a swap of importance between the two variables.

## 8.7

Refer to Exercise 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:

library(missForest)  
library(AppliedPredictiveModeling)  
library(doParallel)  
data("ChemicalManufacturingProcess")  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100)$ximp

## missForest iteration 1 in progress...done!  
## missForest iteration 2 in progress...done!  
## missForest iteration 3 in progress...done!  
## missForest iteration 4 in progress...done!

data.set <- list(org.data.set = chem.imput)

library(tidyverse)  
library(caret)  
library(forecast)  
library(MASS)  
  
trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))

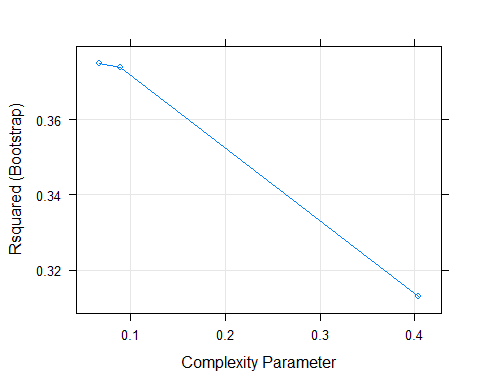
data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))

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

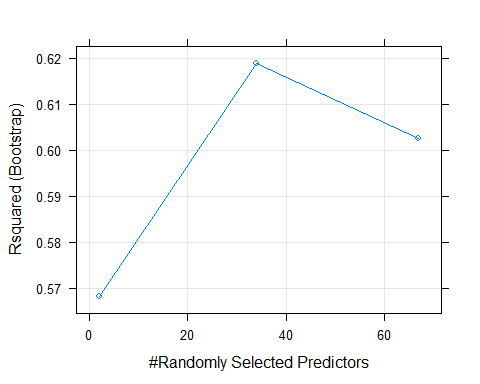
ctrl <- trainControl(method = "boot", number = 15)  
  
rpart.fit <- train(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 method = "rpart",  
 metric = "Rsquared",  
 trControl = ctrl)

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =  
## trainInfo, : There were missing values in resampled performance measures.

plot(rpart.fit)



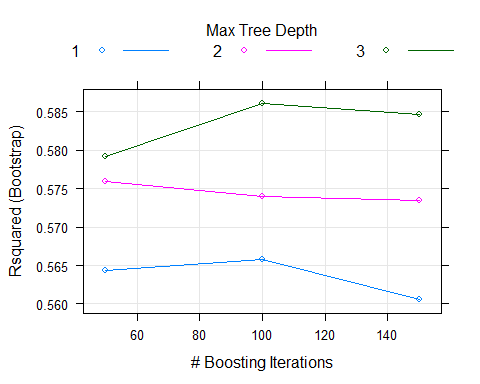
rf.fit <- train(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 method = "rf",  
 metric = "Rsquared",  
 trControl = ctrl,  
 importance = TRUE)  
plot(rf.fit)



gbm.fit <- train(data.set$train %>%   
 dplyr::select(-Yield), data.set$train$Yield,  
 method = "gbm",  
 metric = "Rsquared",  
 trControl = ctrl)

## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0812 nan 0.1000 0.3265  
## 2 2.7358 nan 0.1000 0.2213  
## 3 2.4736 nan 0.1000 0.2279  
## 4 2.2640 nan 0.1000 0.1816  
## 5 2.0698 nan 0.1000 0.1617  
## 6 1.8866 nan 0.1000 0.1125  
## 7 1.7093 nan 0.1000 0.1259  
## 8 1.6250 nan 0.1000 0.0529  
## 9 1.5190 nan 0.1000 0.0830  
## 10 1.4228 nan 0.1000 0.0500  
## 20 0.9278 nan 0.1000 0.0100  
## 40 0.5500 nan 0.1000 -0.0055  
## 60 0.3775 nan 0.1000 -0.0016  
## 80 0.2551 nan 0.1000 -0.0052  
## 100 0.1763 nan 0.1000 -0.0005

plot(gbm.fit)



The best performing model is the random forest with an value at 61.8.

rf.fit$finalModel

##   
## Call:  
## randomForest(x = x, y = y, mtry = param$mtry, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 34  
##   
## Mean of squared residuals: 1.359705  
## % Var explained: 61.12

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?

varImp(rf.fit, scale = FALSE)

## rf variable importance  
##   
## only 20 most important variables shown (out of 67)  
##   
## Overall  
## ManufacturingProcess32 25.016  
## ManufacturingProcess31 10.789  
## ManufacturingProcess17 9.345  
## ManufacturingProcess13 9.003  
## BiologicalMaterial03 8.801  
## BiologicalMaterial06 8.518  
## BiologicalMaterial12 6.293  
## ManufacturingProcess11 6.107  
## ManufacturingProcess36 5.762  
## ManufacturingProcess09 5.365  
## BiologicalMaterial05 5.286  
## BiologicalMaterial02 5.109  
## ManufacturingProcess43 5.104  
## ManufacturingProcess25 5.018  
## BiologicalMaterial11 5.005  
## ManufacturingProcess30 4.926  
## ManufacturingProcess28 4.519  
## BiologicalMaterial09 4.381  
## ManufacturingProcess39 4.354  
## BiologicalMaterial04 4.336

ManufacturingProcess32 is the most important followed by ManufacturingProcess17, BiologicalMaterial03, and ManufacturingProcess13 in that order. The Manufacturing Processes dominate the list and hold the highest importance compared to other variables.

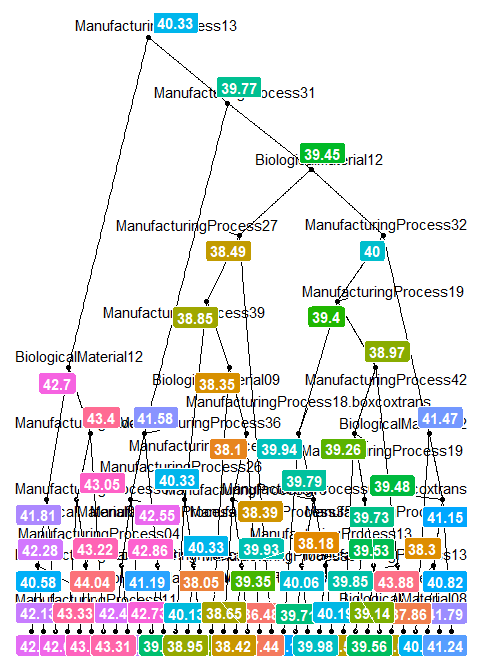
The top 10 lists between the three methods is almost completely different. However, the non-linear method and the forest models both have ManufacturingProcess32 as the most important variable.

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

Random Forest does not natively allow for a plot of this type. I am assuming that the author envisioned another model making the being the best model as my model cannot be used in this way.

Here is a plot of the tree and interestingly we see that the biological process 12 is the first split and then we our manufacturing nodes splitting the tree and the next level.

#https://shiring.github.io/machine\_learning/2017/03/16/rf\_plot\_ggraph  
  
library(dplyr)  
library(ggraph)  
library(igraph)  
  
tree\_func <- function(final\_model,   
 tree\_num) {  
   
 # get tree by index  
 tree <- randomForest::getTree(final\_model,   
 k = tree\_num,   
 labelVar = TRUE) %>%  
 tibble::rownames\_to\_column() %>%  
 # make leaf split points to NA, so the 0s won't get plotted  
 mutate(`split point` = ifelse(is.na(prediction), `split point`, NA))  
   
 # prepare data frame for graph  
 graph\_frame <- data.frame(from = rep(tree$rowname, 2),  
 to = c(tree$`left daughter`, tree$`right daughter`))  
   
 # convert to graph and delete the last node that we don't want to plot  
 graph <- graph\_from\_data\_frame(graph\_frame) %>%  
 delete\_vertices("0")  
   
 # set node labels  
 V(graph)$node\_label <- gsub("\_", " ", as.character(tree$`split var`))  
 V(graph)$leaf\_label <- as.character(round(tree$prediction,2))  
 V(graph)$split <- as.character(round(tree$`split point`, digits = 2))  
   
 # plot  
 plot <- ggraph(graph, 'dendrogram') +   
 theme\_bw() +  
 geom\_edge\_link() +  
 geom\_node\_point() +  
 geom\_node\_text(aes(label = node\_label), na.rm = TRUE, repel = TRUE) +  
 geom\_node\_label(aes(label = split), vjust = 2.5, na.rm = TRUE, fill = "white") +  
 geom\_node\_label(aes(label = leaf\_label, fill = leaf\_label), na.rm = TRUE,   
 repel = TRUE, colour = "white", fontface = "bold", show.legend = FALSE) +  
 theme(panel.grid.minor = element\_blank(),  
 panel.grid.major = element\_blank(),  
 panel.background = element\_blank(),  
 plot.background = element\_rect(fill = "white"),  
 panel.border = element\_blank(),  
 axis.line = element\_blank(),  
 axis.text.x = element\_blank(),  
 axis.text.y = element\_blank(),  
 axis.ticks = element\_blank(),  
 axis.title.x = element\_blank(),  
 axis.title.y = element\_blank(),  
 plot.title = element\_text(size = 18))  
   
 print(plot)  
}  
  
tree\_num <- max(which(rf.fit$finalModel$forest$ndbigtree == min(rf.fit$finalModel$forest$ndbigtree)))  
tree\_func(final\_model = rf.fit$finalModel, tree\_num = tree\_num)



# Recommender Systems

Imagine 10000 receipts sitting on your table. Each receipt represents a transaction with items that were purchased. The receipt is a representation of stuff that went into a customer’s basket - and therefore ‘Market Basket Analysis’.

That is exactly what the Groceries Data Set contains: a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a transaction and each column in a row represents an item.

Here is the dataset = GroceryDataSet.csv (comma separated file)

You assignment is to use R to mine the data for association rules. You should report support, confidence and lift and your top 10 rules by lift.

## Loading the Data

df <- read.transactions("https://raw.githubusercontent.com/ChristopheHunt/MSDA---Coursework/master/Data%20624/Homework%2011/GroceryDataSet.csv", sep = ",", format = c("basket"))

Let’s look at the first 5 transactions to understand how the data is being loaded.

inspect(df[1:5])

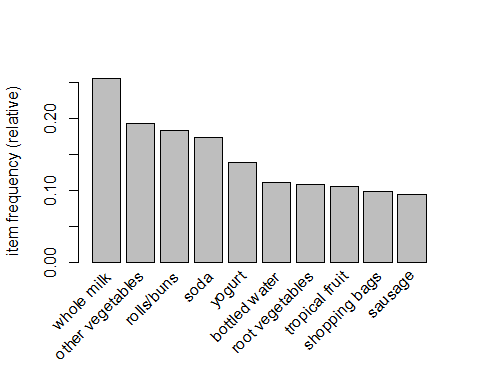
## items   
## [1] {citrus fruit,   
## margarine,   
## ready soups,   
## semi-finished bread}   
## [2] {coffee,   
## tropical fruit,   
## yogurt}   
## [3] {whole milk}   
## [4] {cream cheese,   
## meat spreads,   
## pip fruit,   
## yogurt}   
## [5] {condensed milk,   
## long life bakery product,  
## other vegetables,   
## whole milk}

Now we can review common occurences. We see that vgetables and milk are at the top of the list, which is reassuring.

summary(df)

## transactions as itemMatrix in sparse format with  
## 9835 rows (elements/itemsets/transactions) and  
## 169 columns (items) and a density of 0.02609146   
##   
## most frequent items:  
## whole milk other vegetables rolls/buns soda   
## 2513 1903 1809 1715   
## yogurt (Other)   
## 1372 34055   
##   
## element (itemset/transaction) length distribution:  
## sizes  
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15   
## 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55   
## 16 17 18 19 20 21 22 23 24 26 27 28 29 32   
## 46 29 14 14 9 11 4 6 1 1 1 1 3 1   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 2.000 3.000 4.409 6.000 32.000   
##   
## includes extended item information - examples:  
## labels  
## 1 abrasive cleaner  
## 2 artif. sweetener  
## 3 baby cosmetics

itemFrequencyPlot(df, topN = 10)



Further we can run the apriori function to determine the associated rules. Because we have so many different choices to purchase we will set a lower support. 100 out 9835 gives us a support value of 0.01016777, we will use an arbitrary confidence of .5.

rules <- apriori(df, parameter = list(support = 0.01016777, confidence = 0.5))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support  
## 0.5 0.1 1 none FALSE TRUE 5 0.01016777  
## minlen maxlen target ext  
## 1 10 rules FALSE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 100   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
## sorting and recoding items ... [88 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 done [0.00s].  
## writing ... [14 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

inspect(sort(rules, by = "lift")[1:10])

## lhs rhs support confidence lift count  
## [1] {citrus fruit,   
## root vegetables} => {other vegetables} 0.01037112 0.5862069 3.029608 102  
## [2] {root vegetables,   
## tropical fruit} => {other vegetables} 0.01230300 0.5845411 3.020999 121  
## [3] {rolls/buns,   
## root vegetables} => {other vegetables} 0.01220132 0.5020921 2.594890 120  
## [4] {root vegetables,   
## yogurt} => {other vegetables} 0.01291307 0.5000000 2.584078 127  
## [5] {butter,   
## other vegetables} => {whole milk} 0.01148958 0.5736041 2.244885 113  
## [6] {root vegetables,   
## tropical fruit} => {whole milk} 0.01199797 0.5700483 2.230969 118  
## [7] {root vegetables,   
## yogurt} => {whole milk} 0.01453991 0.5629921 2.203354 143  
## [8] {domestic eggs,   
## other vegetables} => {whole milk} 0.01230300 0.5525114 2.162336 121  
## [9] {whipped/sour cream,   
## yogurt} => {whole milk} 0.01087951 0.5245098 2.052747 107  
## [10] {rolls/buns,   
## root vegetables} => {whole milk} 0.01270971 0.5230126 2.046888 125

Reporting our top rules by lift we see interesting results. It appears that buying vegetables and fruits suggests that similar purchases of other vegetables are possible. We also see those buying butter, eggs, and other animal products are more apt to buy whole milk. The exercise is informative of the power in this type of analysis.