Bin Lin

Homework KJ Chapter 6

6.3. A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is to 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:

a. Start R and use these commands to load the data:

```
library(AppliedPredictiveModeling)
library(dplyr)
library(forecast)
#data(package="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.

b. A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values (e.g., see Sect. 3.8).

Approaches: We can use Mice package to impute the missing values. Mice means multivariate imputation by chained equations. To improve the running time, I only run one time of imputation and use the default function ppm. Ppm means predictive mean matching. " It is similar to the regression method except that for each missing value, it imputes a value randomly from a set of observed values whose predicted values are closest to the predicted value for the missing value from the simulated regression model" (Heitjan and Little 1991; Schenker and Taylor 1996).

Interpretation:

The Mice package is very useful in terms of imputating values. As the end result shows, there is no more missing value in the data frame.

```
library(mice)
a <- mice(ChemicalManufacturingProcess, m = 1, method = "pmm", print = F)
C_M_P <- complete(a)


result<- C_M_P %>%
  select(everything()) %>%
  summarize_all(funs(sum(is.na(.))))


data.frame(sort(result, decreasing = TRUE))
```

c. 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?

Approaches: To administer a series of transformations to multiple data sets, the caret class preProcess has the ability to transform, center, scale, or impute values, as well as excluding "near zero-variance" predictors. The function calculates the required quantities for the transformation. After calling the preProcess function, the predict method applies the results to a set of data.

```r
library(e1071)

library(caret)

CMP_trans <- preProcess(C_M_P, method = c("nzv", "BoxCox", "center",
"scale"))

transformed <- predict(CMP_trans, C_M_P)
```

Approaches: The base R function sample can create simple random splits of the data. To create stratified random splits of the data (based on the classes), the createDataPartition function in the caret package can be used. The percent of data that will be allocated to the training set should be specified.

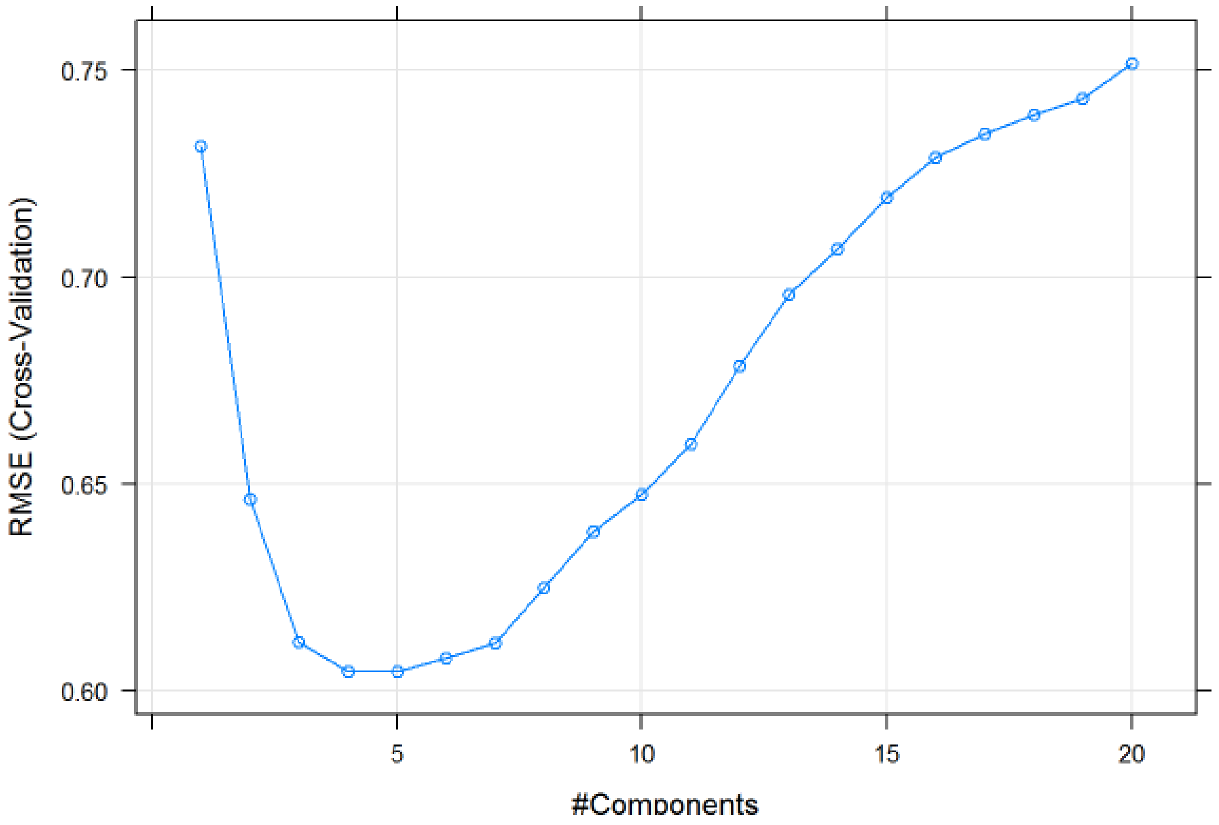
```r
set.seed(88)

trainingRows <- createDataPartition(transformed$Yield, p = .80, list= FALSE)


yield_train <- transformed[trainingRows, 1]

predictor_train <- transformed[trainingRows, -1]


yield_test <- transformed[-trainingRows, 1]

predictor_test <- transformed[-trainingRows, -1]
```

Approaches: The tune function of the e1071 package can determine parameter settings using resampling. The train function generates a resampling estimate of performance. Repeated 10-fold cross-validation can be specified with the trainControl function. tuneLength set to the maximum number of dimensions that we want to evaluate.

```r
library(pls)

set.seed(888)


ctrl <- trainControl(method = "cv", number = 10)

plsTune <- train(predictor_train, yield_train, method = "pls", tuneLength =
20, trControl = ctrl)
```

The plot method can be used to visualize the performance profile. RMSE was used to select the optimal model using the smallest value. The optimal model contains 5 components.

```
plot(plsTune)
```



d. 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?

Interpretation: To compute the model product yield values for new samples, the predict method is used. Then use the caret function defaultSummary to estimate the test set performance, The result ends up to be 0.6695361. On the other hand, the resampled performance metric on the training set is 0.6080308. They are very closed to each other.

```
CMP_Prediction <- predict(plsTune, predictor_test, ncomp = 1:5)
results <- data.frame(obs = yield_test, pred = CMP_Prediction)
defaultSummary(results)
##      RMSE  Rsquared       MAE
## 0.6654727 0.5944409 0.5251922
```

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

Approaches: There are two functions that estimate the importance of each predictor in the model: evimp in the earth package and varImp in the caret package. We are going to use varImp.

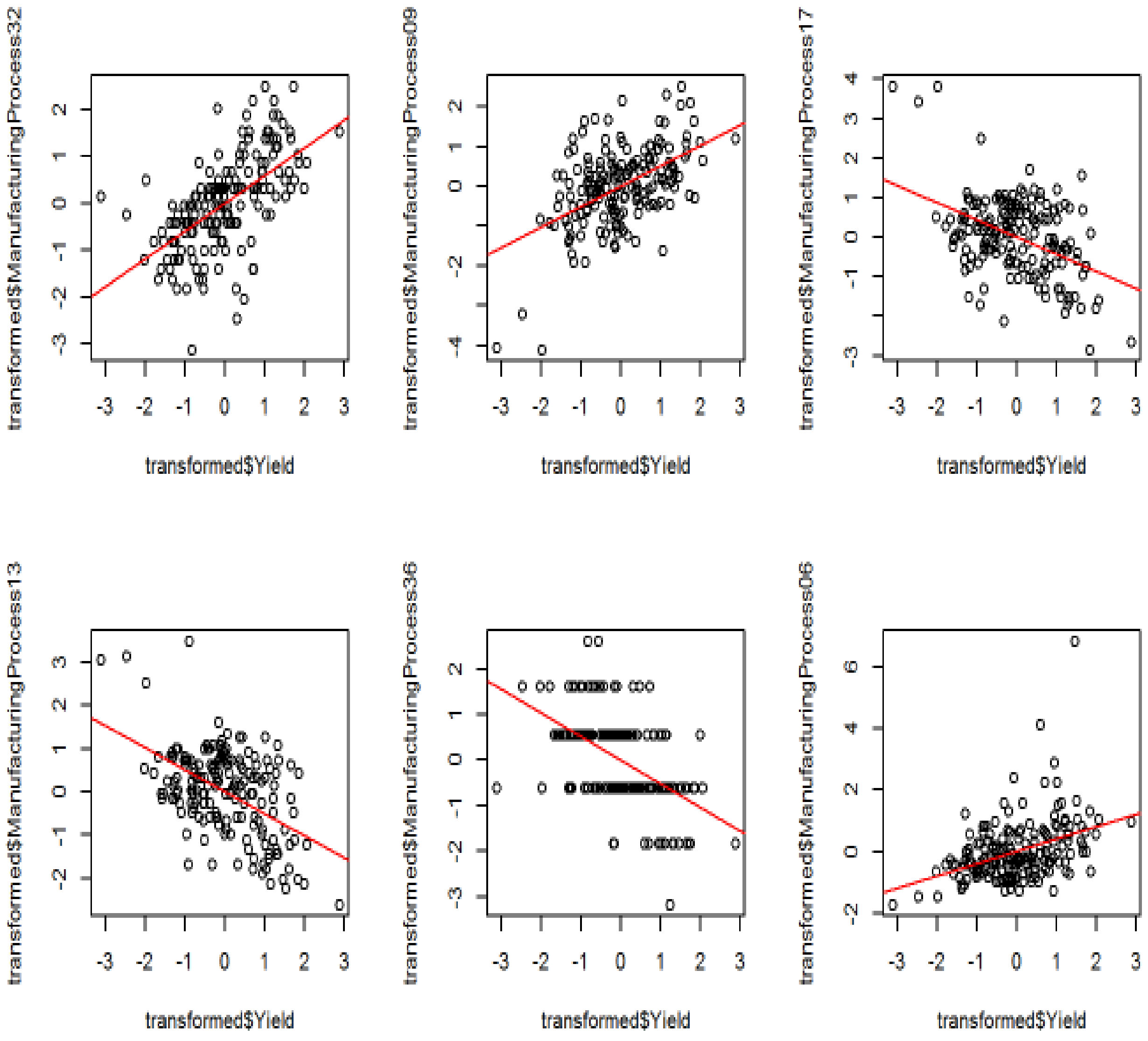
Interpretation: According to the following list, ManufacturingProcess32 is the most important in the model. Apparently, process predictors dominate the list, because they take up the top 5 positions.

```
varImp(plsTune)
## pls variable importance
##
##   only 20 most important variables shown (out of 56)
##
##                        Overall
## ManufacturingProcess32  100.00
## ManufacturingProcess09   87.42
## ManufacturingProcess17   81.64
## ManufacturingProcess13   81.14
## ManufacturingProcess36   75.75
## ManufacturingProcess06   67.47
## BiologicalMaterial02     55.59
## BiologicalMaterial06     55.57
## ManufacturingProcess33   53.09
## ManufacturingProcess34   52.61
## ManufacturingProcess11   51.26
## BiologicalMaterial12     51.09
## BiologicalMaterial03     51.01
## BiologicalMaterial08     48.46
## BiologicalMaterial11     48.44
## BiologicalMaterial04     48.27
## ManufacturingProcess12   45.51
## BiologicalMaterial01     42.44
## ManufacturingProcess37   40.82
## ManufacturingProcess28   37.43
```

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

Interpretation: Some of the predictors are positively correlated with the response, while the others are negatively correlated. The information can be helpful because if we improve the predictors with positive correlation, the response will likely get improved and vice versa.

```r
par(mfrow = c(2, 3))
plot(transformed$Yield, transformed$ManufacturingProcess32)
abline(lm(transformed$Yield~transformed$ManufacturingProcess32), col="red")

plot(transformed$Yield, transformed$ManufacturingProcess09)
abline(lm(transformed$Yield~transformed$ManufacturingProcess09), col="red")

plot(transformed$Yield, transformed$ManufacturingProcess17)
abline(lm(transformed$Yield~transformed$ManufacturingProcess17), col="red")

plot(transformed$Yield, transformed$ManufacturingProcess13)
abline(lm(transformed$Yield~transformed$ManufacturingProcess13), col="red")

plot(transformed$Yield, transformed$ManufacturingProcess36)
abline(lm(transformed$Yield~transformed$ManufacturingProcess36), col="red")

plot(transformed$Yield, transformed$ManufacturingProcess06)
abline(lm(transformed$Yield~transformed$ManufacturingProcess06), col="red")
```


transformed$ManufacturingProcess32
transformed$ManufacturingProcess09
transformed$ManufacturingProcess17
transformed$ManufacturingProcess13
transformed$ManufacturingProcess36
transformed$ManufacturingProcess06
transformed$Yield

Homework KJ Chapter 7

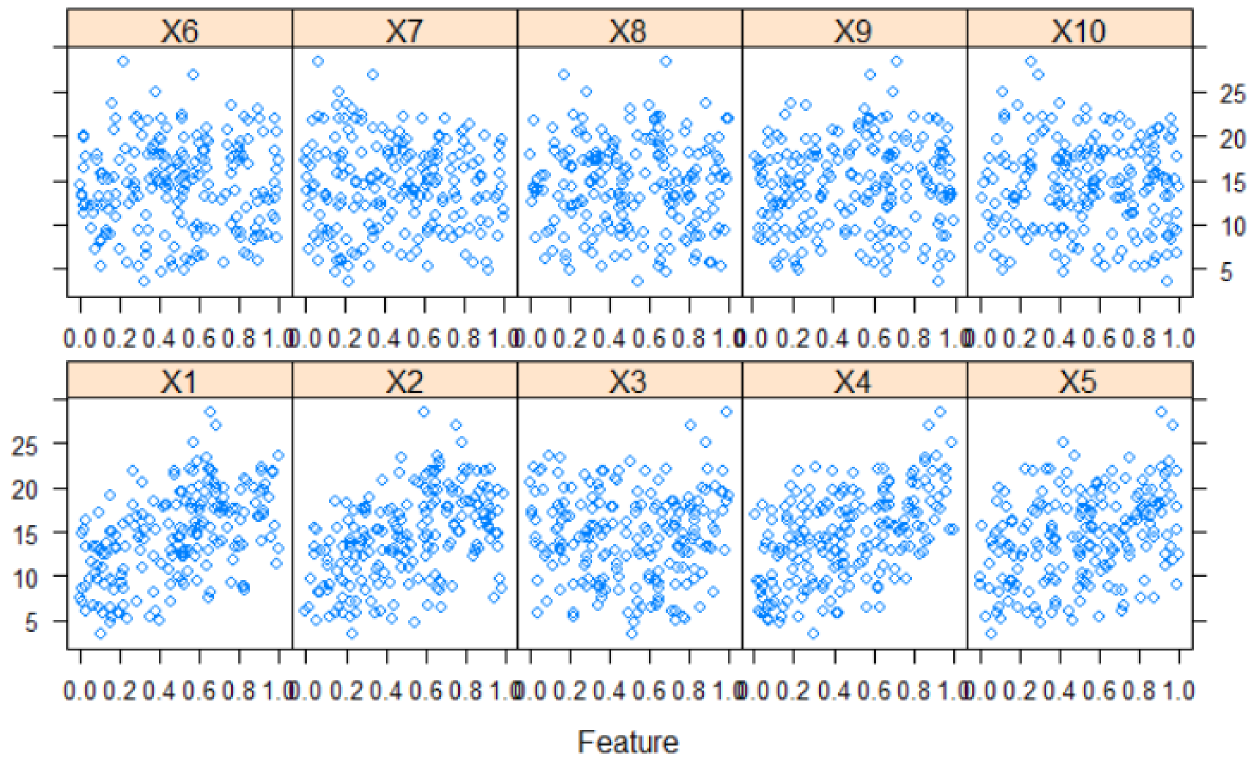
7.2. Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data:

$$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + N(0, \sigma^2)$$

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

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

```r
library(mlbench)
library(caret)
library(earth)
library(pls)
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)
trainingData$x <- data.frame(trainingData$x)
featurePlot(trainingData$x, trainingData$y)
```



```r
testData <- mlbench.friedman1(5000, sd = 1)

testData$x <- data.frame(testData$x)

knnModel <- train(x = trainingData$x, y = trainingData$y, method = "knn",
preProc = c("center", "scale"), tuneLength = 10)

knnModel
```

```
## k-Nearest Neighbors
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  3.565620  0.4887976  2.886629
##    7  3.422420  0.5300524  2.752964
##    9  3.368072  0.5536927  2.715310
##   11  3.323010  0.5779056  2.669375
##   13  3.275835  0.6030846  2.628663
##   15  3.261864  0.6163510  2.621192
##   17  3.261973  0.6267032  2.616956
##   19  3.286299  0.6281075  2.640585
##   21  3.280950  0.6390386  2.643807
##   23  3.292397  0.6440392  2.656080
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 15.
```

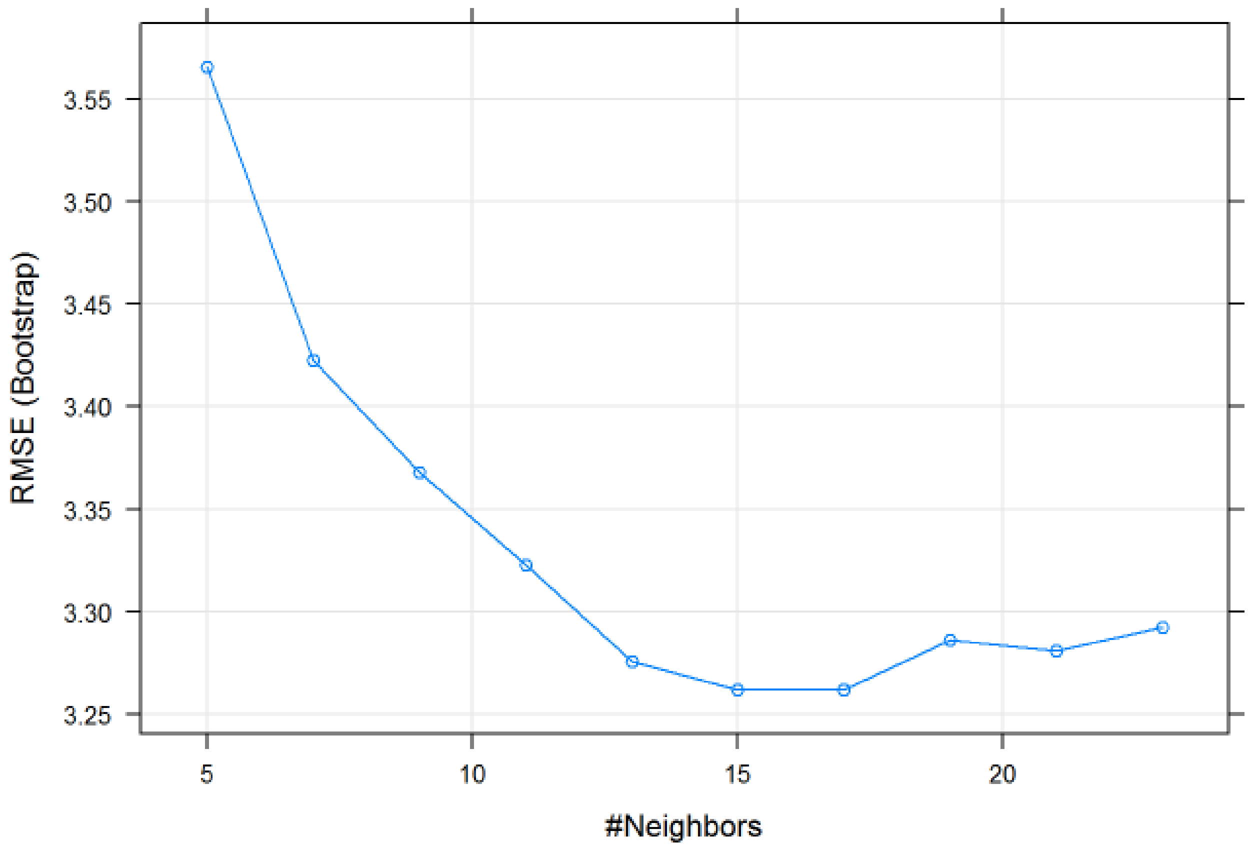
```r
knnPred <- predict(knnModel, newdata = testData$x)
```

```
postResample(pred = knnPred, obs = testData$y)
```

```
##      RMSE  Rsquared       MAE
## 3.1750657 0.6785946 2.5443169
```

Interpretation: According to the following graph, the optimal k-Nearest Neighbors model contains 15 neighbors with RMSE value at 3.175.

```
plot(knnModel)
```



Approaches: The following code, the data was used to train for MARS model, after the preprocess procedures such as center, scale have already completed.

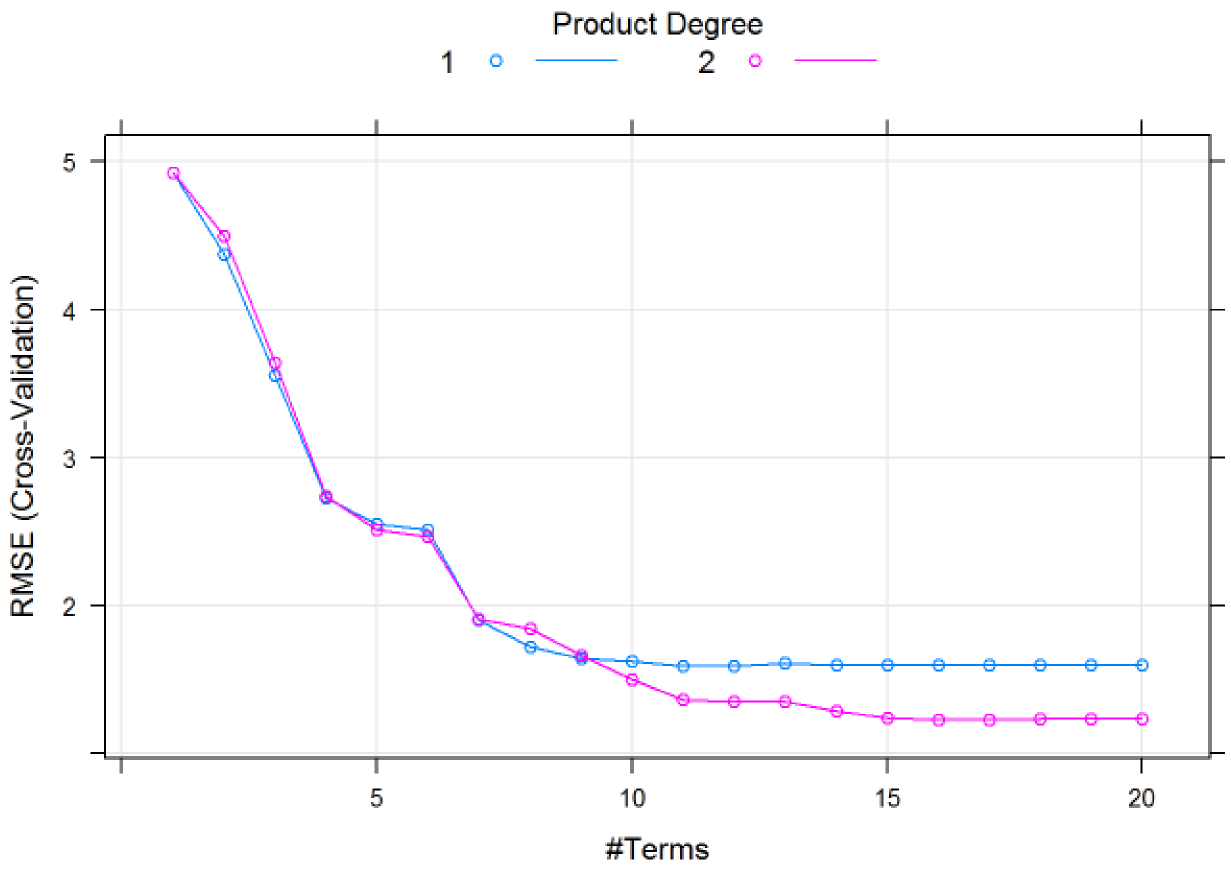
```
set.seed(888)

marsGrid <- expand.grid(degree = 1:2, nprune = 1:20)

marsTuned <- train(x = trainingData$x, y = trainingData$y, method = "earth",
tuneGrid = marsGrid, trControl = trainControl(method = "cv"), preProc =
c("center", "scale"))
```

Interpretation: MARS model is apparently better than KNN model since its RMSE value is 1.323, which is much less than the one generated from KNN model. The final values used for the model were nprune = 16 and degree = 2.In addition, MARS selects the informative predictors (X1-X5) only. X6-X10 predictors have no importance at all as shown on the following earth variable importance list.

```
marsPred <- predict(marsTuned, newdata = testData$x)
postResample(pred = marsPred, obs = testData$y)
##      RMSE  Rsquared       MAE
## 1.2793868 0.9343367 1.0091132
```

```
varImp(marsTuned)
## earth variable importance
##
##     Overall
## X1   100.00
## X4    85.12
## X2    69.20
## X5    49.23
## X3    39.89
## X8     0.00
## X6     0.00
## X9     0.00
## X10    0.00
## X7     0.00
```

```
plot(marsTuned)
```



```
marsTuned
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        1      4.921307        NaN  4.0422513
##   1        2      4.372600  0.2494859  3.6584051
##   1        3      3.555270  0.4985649  2.8726615
##   1        4      2.726352  0.7089775  2.1900423
##   1        5      2.547907  0.7479963  2.0594070
##   1        6      2.509322  0.7615072  1.9793715
##   1        7      1.901733  0.8634079  1.4904166
##   1        8      1.719886  0.8835157  1.3687368
##   1        9      1.644159  0.8921695  1.3043667
##   1       10      1.623350  0.8948379  1.2795465
##   1       11      1.589075  0.9002714  1.2555563
##   1       12      1.593492  0.8997416  1.2497490
##   1       13      1.610554  0.8972277  1.2600103
##   1       14      1.600163  0.8985099  1.2527223
##   1       15      1.600163  0.8985099  1.2527223
##   1       16      1.600163  0.8985099  1.2527223
##   1       17      1.600163  0.8985099  1.2527223
##   1       18      1.600163  0.8985099  1.2527223
##   1       19      1.600163  0.8985099  1.2527223
##   1       20      1.600163  0.8985099  1.2527223
##   2        1      4.921307        NaN  4.0422513
##   2        2      4.496073  0.2034636  3.7413677
##   2        3      3.638997  0.4724445  2.9952571
##   2        4      2.741191  0.7116752  2.2002016
##   2        5      2.513763  0.7579066  2.0381846
##   2        6      2.465818  0.7681812  1.9718287
##   2        7      1.911666  0.8594136  1.5023921
##   2        8      1.846490  0.8631160  1.3941037
##   2        9      1.660966  0.8904560  1.3065837
##   2       10      1.498962  0.9085211  1.1720222
##   2       11      1.366536  0.9224437  1.0776368
##   2       12      1.354300  0.9255822  1.0697916
##   2       13      1.349547  0.9253808  1.0753422
##   2       14      1.286753  0.9330843  1.0312404
##   2       15      1.238795  0.9377730  0.9894584
##   2       16      1.226555  0.9391833  0.9784266
##   2       17      1.230141  0.9391998  0.9815644
##   2       18      1.234600  0.9388660  0.9846447
##   2       19      1.234600  0.9388660  0.9846447
##   2       20      1.234600  0.9388660  0.9846447
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 16 and degree = 2.
```

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.

a. Which nonlinear regression model gives the optimal resampling and test set performance?

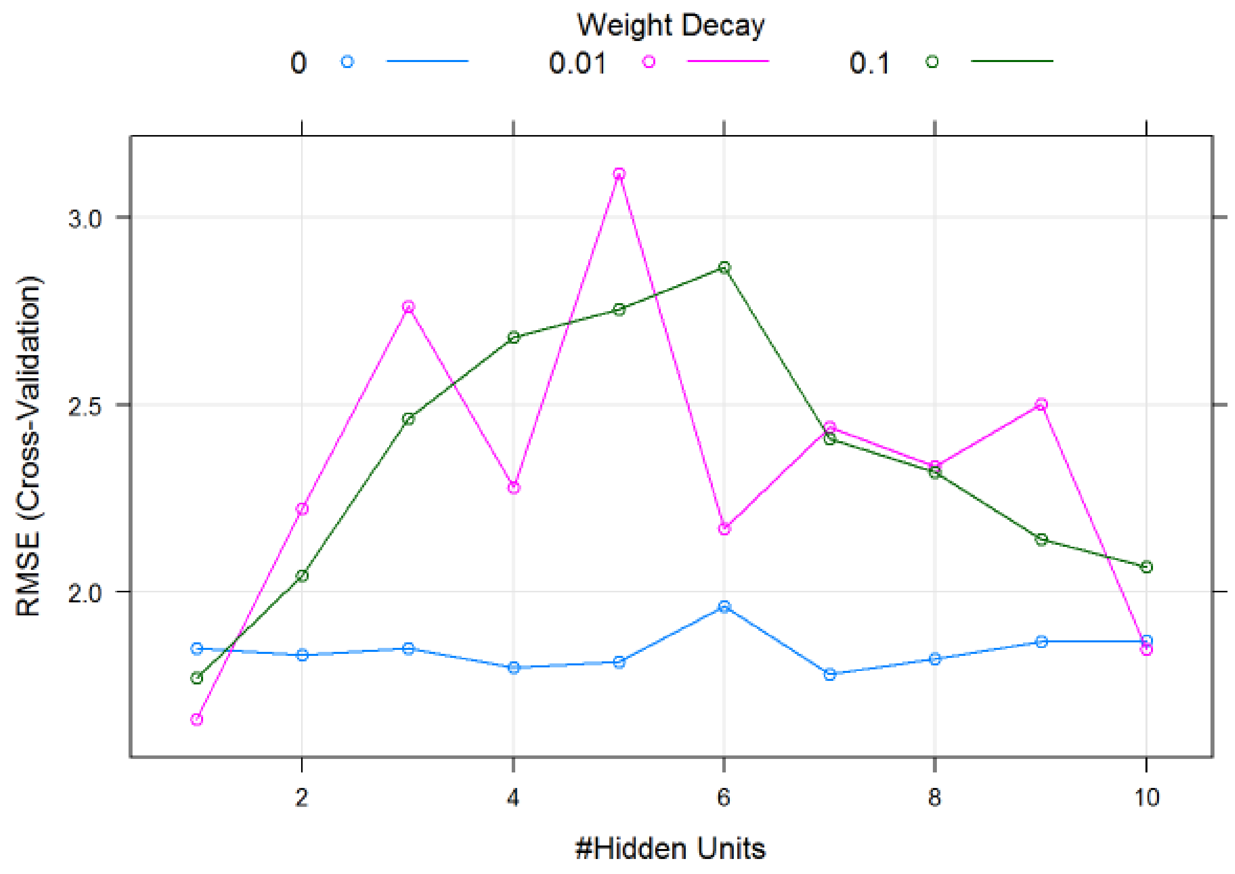
```r
library(AppliedPredictiveModeling)
library(mice)
library(e1071)
#install.packages("RANN")
library(RANN)

data(ChemicalManufacturingProcess)
set.seed(888)

a <- mice(ChemicalManufacturingProcess, m = 1, method = "pmm", print = F)
C_M_P <- complete(a)

trainingRows <- createDataPartition(C_M_P$Yield, p = .80, list= FALSE)

yield_train <- C_M_P[trainingRows, 1]
predictor_train <- C_M_P[trainingRows, -1]

yield_test <- C_M_P[-trainingRows, 1]
predictor_test <- C_M_P[-trainingRows, -1]

CMP_trans <- preProcess(predictor_train, method = c("nzv", "BoxCox",
"center", "scale", "knnImpute"))
ctrl <- trainControl(method = "cv", number = 10)
```

Neural Networks:

```
set.seed(888)


nnetGrid <- expand.grid(decay = c(0, 0.01, .1), size = c(1:10))


nnetTune <- train(x = predictor_train, y = yield_train, method = "nnet",
tuneGrid = nnetGrid, trControl = ctrl, linout = TRUE, trace = FALSE, maxit =
500)

nnetPred <- predict(nnetTune, newdata = predictor_test)

postResample(pred = nnetPred, obs = yield_test)
```

```
##      RMSE  Rsquared       MAE
## 1.6052444 0.3915792 1.3444695
```

```
plot(nnetTune)
```

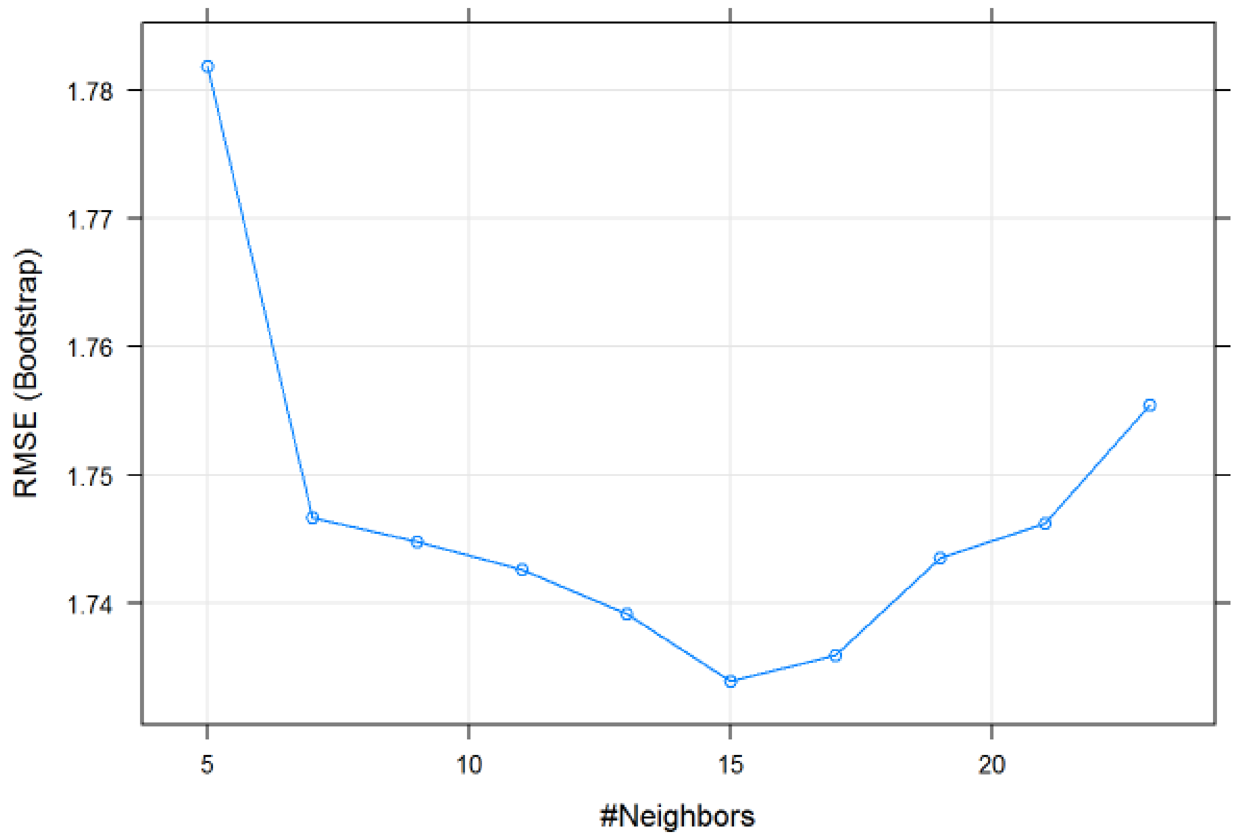


K-Nearest Neighbors:

```
set.seed(888)

knnTune <- train(x = predictor_train, y = yield_train, method = "knn",
tuneLength = 10)


knnPred <- predict(knnTune, newdata = predictor_test)

postResample(pred = knnPred, obs = yield_test)
```

```
##      RMSE  Rsquared       MAE

## 1.1436604 0.7049123 0.9106875
```

```
plot(knnTune)
```



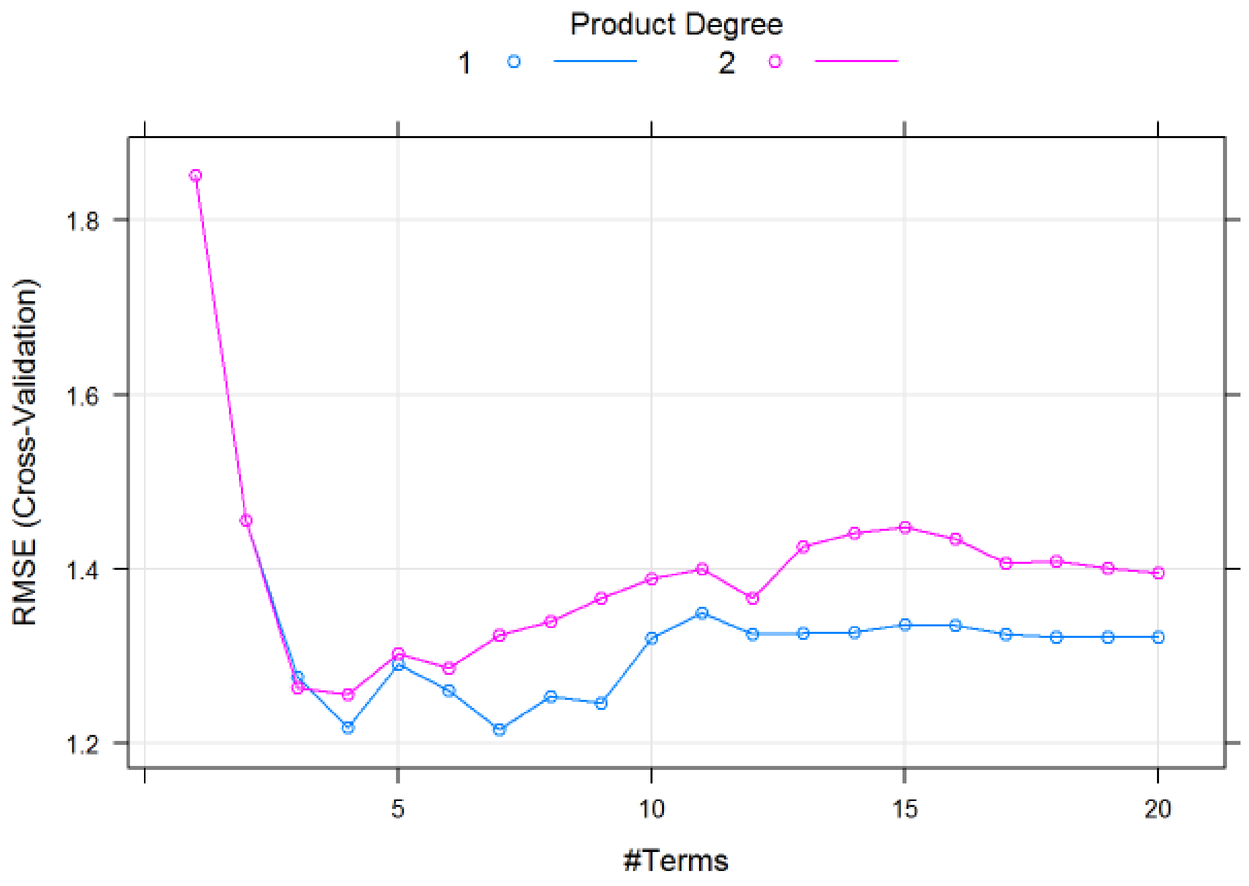
Multivariate Adaptive Regression Splines

```
set.seed(888)
```

```
marsGrid <- expand.grid(degree = 1:2, nprune = 1:20)

marsTune <- train(x = predictor_train, y = yield_train, method = "earth",
tuneGrid = marsGrid, trControl = ctrl)

marsPred <- predict(marsTune, newdata = predictor_test)

postResample(pred = marsPred, obs = yield_test)
```

```
##      RMSE  Rsquared       MAE

## 1.0643609 0.6254264 0.9566682
```

```
plot(marsTune)
```



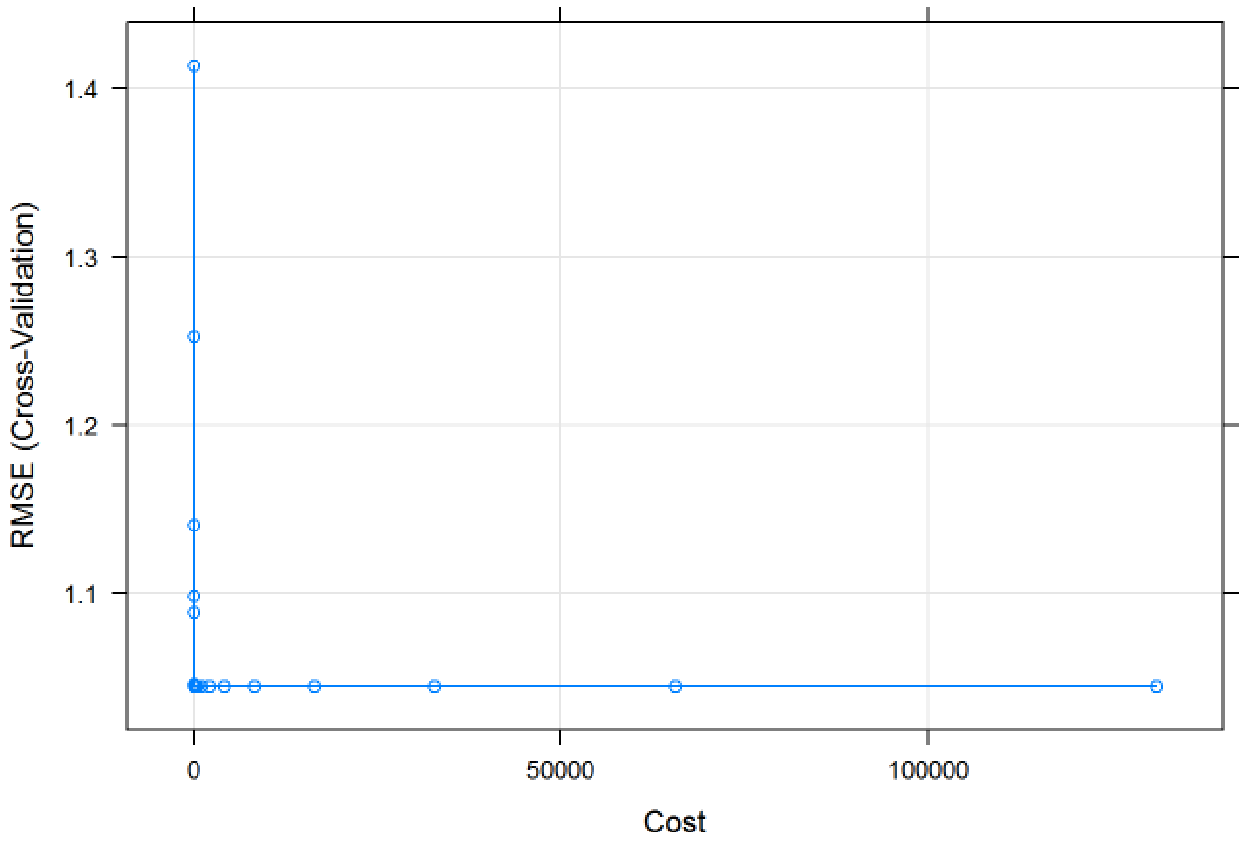
Support Vector Machines:

```
set.seed(888)
```

```
svmRTune <- train(predictor_train, yield_train, method = "svmRadial",
tuneLength = 20, trControl = ctrl)

svmRPred <- predict(svmRTune, newdata = predictor_test)

postResample(pred = svmRPred, obs = yield_test)
```

```
##      RMSE  Rsquared       MAE
## 1.0441130 0.6537003 0.8766606
```

```
plot(svmRTune)
```



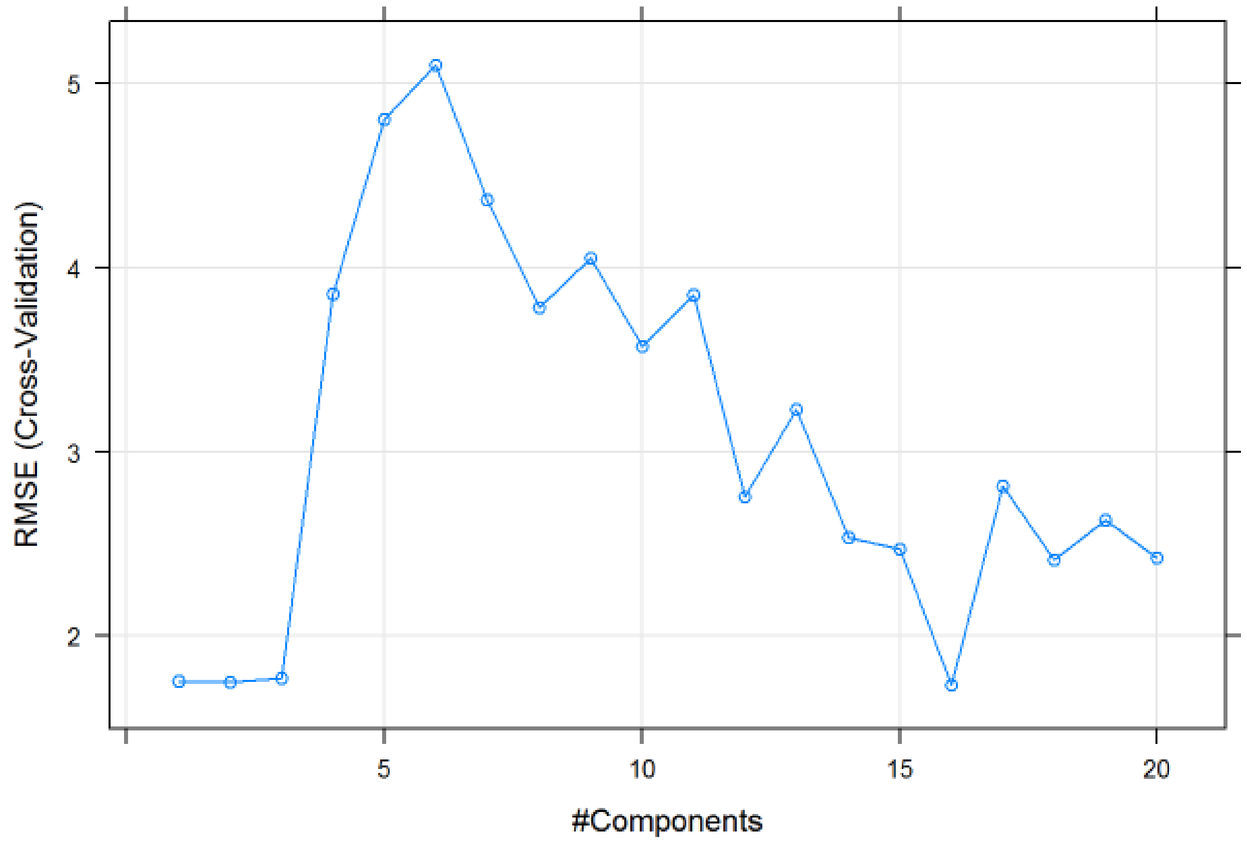
Partial Least Squares:

```
set.seed(888)
```

```
plsTune <- train(predictor_train, yield_train, method = "pls", tuneLength =
20, trControl = ctrl)


plsPred <- predict(plsTune, newdata = predictor_test)

postResample(pred = plsPred, obs = yield_test)
```

```
##      RMSE  Rsquared       MAE

## 2.7779022 0.2398376 1.3671123
```

```
plot(plsTune)
```



```
pls_RMSE <- min(plsTune$results$RMSE)

nnet_RMSE <- min(nnetTune$results$RMSE)

mars_RMSE <- min(marsTune$results$RMSE)

svmR_RMSE <- min(svmRTune$results$RMSE)

knn_RMSE <- min(knnTune$results$RMSE)
```

```
result <- data.frame(Model = c("pls", "nnet", "mars", "svmR", "knn"), RMSE =
c(pls_RMSE, nnet_RMSE, mars_RMSE, svmR_RMSE, knn_RMSE))


result
```

```
##   Model     RMSE
## 1   pls 1.732079
## 2  nnet 1.660641
## 3  mars 1.215567
## 4  svmR 1.044491
## 5   knn 1.733900
```

Interpretation: Based on the result that I obtained for each model, the best model that generate optimal resampling and test set performance is the Support Vector Machines model using the method “svmRadial”. Because this model has the lowest RMSE value which is 1.044.

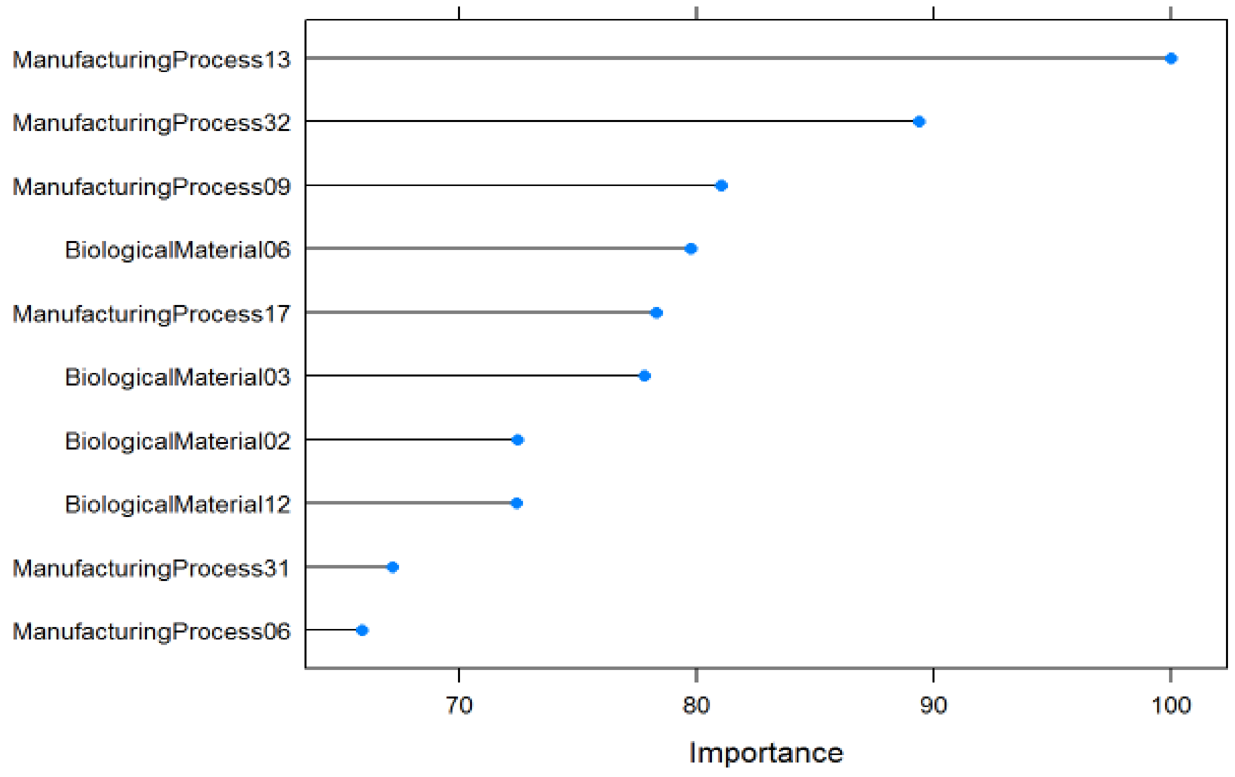
b. Which predictors are most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

```
varImp(svmRTune)
```

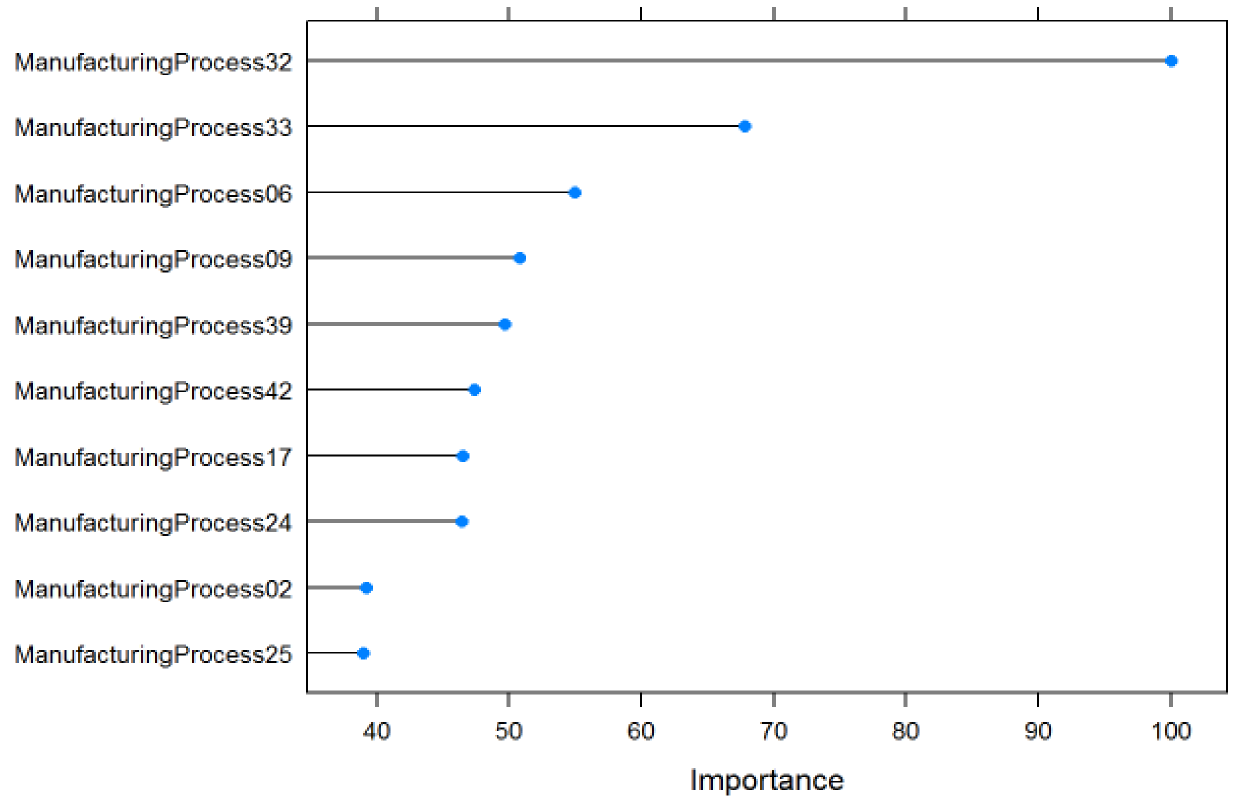
```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 57)
##
##                          Overall
## ManufacturingProcess13  100.00
## ManufacturingProcess32   89.39
## ManufacturingProcess09   81.01
## BiologicalMaterial06     79.74
## ManufacturingProcess17   78.30
## BiologicalMaterial03     77.77
## BiologicalMaterial02     72.42
## BiologicalMaterial12     72.38
## ManufacturingProcess31   67.15
## ManufacturingProcess06   65.88
## BiologicalMaterial11     61.96
## ManufacturingProcess36   60.51
## ManufacturingProcess30   48.66
## BiologicalMaterial08     48.35
## BiologicalMaterial04     45.39
## ManufacturingProcess33   43.97
## ManufacturingProcess29   43.70
## BiologicalMaterial01     42.25
## ManufacturingProcess11   38.09
## BiologicalMaterial09     37.04
```

Interpretation: By comparing the importance figure between the optimal non-linear model and the optimal linear model, we are able know ManufacturingProcess32, 06, and 09 all have significant weights on both model. PLS model is solely built upon manufacturing process predictors. However, SVM model gives more weights to Biological Material, when 4 out of top 10 predictors are from this class. The most surprising finding is that ManufacturingProcess13 is the most important predictor at SVM model, but it is not one of the top 10 predictors for PLS model.

```
par(mfrow = c(1, 2))
plot(varImp(svmRTune), top = 10)
```



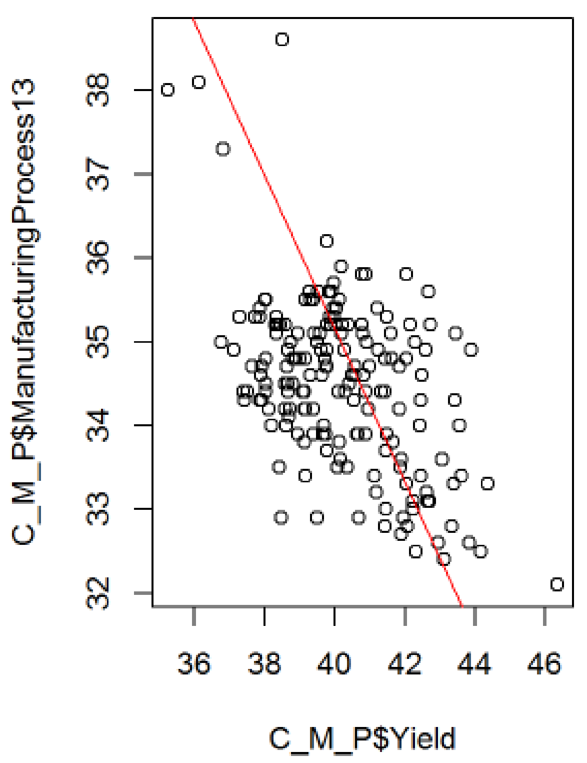
```
plot(varImp(plsTune), top = 10)
```

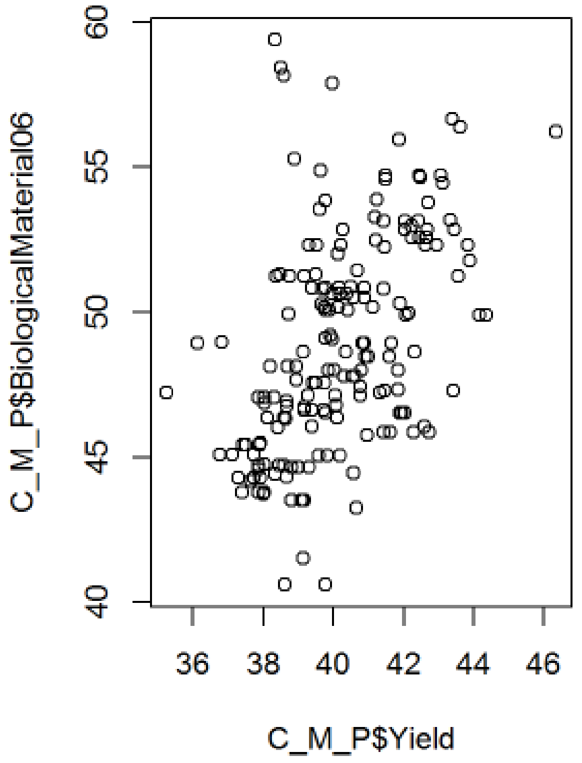


c. Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

Interpretation: Out of the top 5 predictors, SVM model has large weight on ManufacturingProcess13 and BiologicalMaterial06, which are unique to this model. Therefore, I am going to investigate on the relationship between these two predictors with their respective yield in particular. The following graph proves that the yield has negative relationship with ManufacturingProcess13 and positive relationship with BiologicalMaterial06.

```
par(mfrow = c(1, 2))
plot(C_M_P$Yield, C_M_P$ManufacturingProcess13)
abline(lm(C_M_P$Yield ~ C_M_P$ManufacturingProcess13), col="red")


plot(C_M_P$Yield, C_M_P$BiologicalMaterial06)
abline(lm(C_M_P$Yield ~ C_M_P$BiologicalMaterial06), col="red")
```





Homework KJ Chapter 8

8.1. Recreate the simulated data from Exercise 7.2:

```r
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:

```r
library(randomForest)
library(caret)
```

```r
set.seed(100)

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)?

Interpretation: The random forest model did not significantly use the uninformative predictors(V6-V10) due to their importance level are much smaller than the informative predictors(V1-V5).

```r
rfImp1
##         Overall
## V1   8.63576145
## V2   6.36633951
## V3   0.73342992
## V4   7.72426144
## V5   2.28781522
## V6   0.04503439
## V7   0.01582424
## V8  -0.10274799
## V9  -0.07855803
## V10 -0.05070734
```

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

```
simulated2 <- simulated
simulated2$V11 <- simulated$V1 + rnorm(200) * .1
cor(simulated2$V11, simulated2$V1)
## [1] 0.9414719
```

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?

```
set.seed(100)
model2 <- randomForest(y ~ ., data = simulated2, importance = TRUE, ntree =
1000)
rfImp2 <- varImp(model2, scale = FALSE)
rfImp2
##         Overall
## V1   5.73035635
## V2   6.33253246
## V3   0.64453732
## V4   6.75632730
## V5   2.08424598
## V6   0.14382046
## V7   0.01590056
## V8   0.01734896
## V9  -0.08433143
## V10 -0.05625675
## V11  4.19983475
```

Interpretation: The importance score for V1 change from 8.64 to 5.73. After I added 11th predictor that is also highly correlated with V1, all the informative predictors have decreasing importance levels. For uninformative predictors the changes are not obvious. In addition to that, the 11th predictor also gains high importance level just like V1.

```
rfImp1 <- rbind(rfImp1, NA)
cbind(rfImp1, rfImp2)
##         Overall     Overall
## V1   8.63576145  5.73035635
## V2   6.36633951  6.33253246
## V3   0.73342992  0.64453732
## V4   7.72426144  6.75632730
## V5   2.28781522  2.08424598
## V6   0.04503439  0.14382046
## V7   0.01582424  0.01590056
## V8  -0.10274799  0.01734896
## V9  -0.07855803 -0.08433143
## V10 -0.05070734 -0.05625675
## 11           NA  4.19983475
```

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 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)
set.seed(100)


ctrl1 <- cforest_control(mtry = ncol(simulated) - 1)
tree1 <- cforest(y ~ ., data = simulated, controls = ctrl1)
cfimp1 <- varImp(tree1)


ctrl2 <- cforest_control(mtry = ncol(simulated2) - 1)
tree2 <- cforest(y ~ ., data = simulated2, controls = ctrl2)
cfimp2 <- varImp(tree2)
```

Interpretation: The importances show the same pattern as the traditional random forest model. The importance score for V1 change from 9.844 to 3.608. After I added 11th predictor that is also highly correlated with V1, all the informative predictors have decreasing importance levels. For uninformative predictors the changes are not obvious. In addition to that, the 11th predictor also gains high importance level.

```
cfimp1 <- rbind(cfimp1, NA)
cbind(cfimp1, cfimp2)
```

```
##           Overall      Overall
## V1    9.910468369  7.548067198
## V2    8.073769074  7.825289642
## V3    0.008692455 -0.012950356
## V4   10.183786204 10.462213365
## V5    2.353049419  2.327029283
## V6    0.023534821  0.003602874
## V7    0.103887618  0.035647509
## V8   -0.011166128 -0.018521013
## V9   -0.027114637 -0.025849459
## V10   0.028900824 -0.012828186
## 11             NA  2.305285264
```

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

   Interpretation: The same pattern occur for Bagged Trees model, Boosted Trees model, and Cubist Model. The only exception is the 11th predictor on Cubist model. It still has value of zero, unlike the other models, which receive some importance level.

Bagged Trees:

```
library(ipred)
set.seed(100)

baggedTree1 <- bagging(y ~ ., data = simulated)
btimp1 <- varImp(baggedTree1)

baggedTree2 <- bagging(y ~ ., data = simulated2)
btimp2 <- varImp(baggedTree2)
```

```
btimp1 <- rbind(btimp1, NA)
cbind(btimp1, btimp2)
```

```
##        Overall   Overall
## V1   1.6784469 1.7033981
## V10  0.8333179 0.7064733
## V2   1.9844894 1.5559749
## V3   1.2248918 2.4104690
## V4   2.7076782 1.1620767
## V5   2.3904283 2.6479644
## V6   1.0202490 2.4559044
## V7   0.9756717 0.8106956
## V8   0.4303034 1.0994138
## V9   0.6979716 0.6422689
## 11          NA 0.6518883
```

Boosted Trees:

```
library(gbm)
set.seed(100)

gbmModel1 <- gbm(y ~ ., data = simulated, distribution = "gaussian", n.trees
= 100)
gbmimp1 <- varImp(gbmModel1, numTrees = 100)

gbmModel2 <- gbm(y ~ ., data = simulated2, distribution = "gaussian", n.trees
= 100)
gbmimp2 <- varImp(gbmModel2, numTrees = 100)
```

```
gbmimp1 <- rbind(gbmimp1, NA)
cbind(gbmimp1, gbmimp2)
```

```
##       Overall    Overall
## V1   42254.18 25860.3476
## V2   16505.95 19453.3830
## V3       0.00     0.0000
## V4   14225.63 13699.5838
## V5       0.00   811.2224
## V6       0.00     0.0000
## V7       0.00     0.0000
## V8       0.00      0.0000
## V9       0.00      0.0000
## V10      0.00      0.0000
## 11         NA 13398.7897
```

Cubist:

```
library(Cubist)

set.seed(100)
cubist1 <- cubist(simulated[,1:10], simulated$y, committees = 100)
cubimp1 <- varImp(cubist1)

cubist2 <- cubist(simulated2[,c(1:10,12)], simulated$y, committees = 100)
cubimp2 <- varImp(cubist2)
```

```
cubimp1 <- rbind(cubimp1, NA)
cbind(cubimp1, cubimp2)
```

```
##     Overall Overall
## V1     71.5    70.5
## V3     47.0    44.5
## V2     58.5    56.5
## V4     48.0    48.0
## V5     33.0    33.0
## V6     13.0     2.5
## V7      0.0     1.5
## V8      0.0     0.0
## V9      0.0     0.0
## V10     0.0     0.0
## 11       NA     0.0
```

8.2. Use a simulation to show tree bias with different granularities.

Approaches: Trees suffer from selection bias: predictors with a higher number of distinct values are favored over more granular predictors. According to what Loh and Shih said: “The danger occurs when a data set consists of a mix of informative and noise variables, and the noise variables have many more splits than the informative variables. Then there is a high probability that the noise variables will be chosen to split the top nodes of the tree. Pruning will produce either a tree with misleading structure or no tree at all.” In addition, the more missing values, the more biased the selection of predictors.

Interpretation: The informative predictor in the following code has lower variance than the noise predictor. In addition, informative predictor also has higher correlation with the response predictor than the noise predictor. However, the noise predictor has higher importance than informative predictor.

```r
library(rpart)
set.seed(100)

informative <- rep(1:5, 20)
noise <- rnorm(100, mean = 0, sd = 10)
y <- informative + rnorm(100, mean = 0, sd = 5)
df1 <- data.frame(informative, noise, y)

df2 = data.frame(SD = c(sd(informative), sd(noise)), Correlation =
c(cor(informative, y), cor(noise, y)))
row.names(df2) <- c("Informative Predictors", "Noise Predictors")
df2
```

```
##                               SD Correlation
## Informative Predictors  1.421338   0.4851920
## Noise Predictors       10.207104  -0.1512738
```

```r
rpartTree <- rpart(y ~ ., data = df1)
varImp(rpartTree)
```

```
##               Overall
## informative 0.5559885
## noise       0.7086826
```

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:

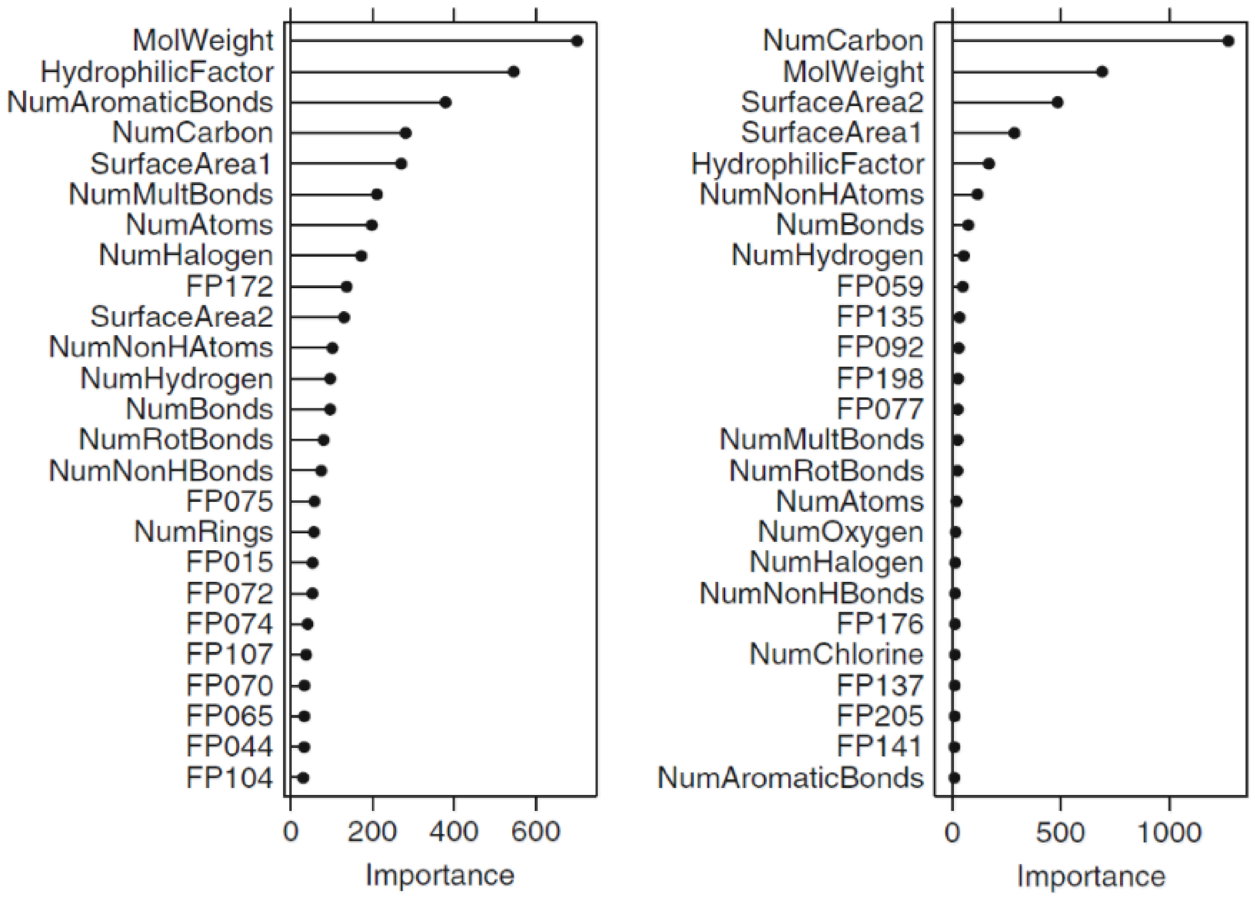


Figure 1

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?

Interpretation: According to "gbm" package documentation, shrinkage is a "parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction." Therefore, the higher the shrinkage value, higher the learning rate, the fewer predictors that it focuses on.

bag.fraction is the "fraction of the training set observations randomly selected to propose the next tree in the expansion. This introduces randomness into the model fit. If bag.fraction<1 then running the same model twice will result in similar but different fits. gbm uses the R random number generator, so set.seed can ensure that the model can be reconstructed. Preferably, the user can save the returned gbm.object using save." Higher bag.fraction values, less randomnesses that the model creates, therefore, fewer predictors that it included in its model.

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

Interpretation: The model on the left would be more predictive of other samples, because it focuses on more predictors than the other models. The model that focuses on less predictors is greedier to run the model faster, however, it ignores the importance of certain predictors.

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

Interpretation: interaction.depth is the maximum depth of variable interactions. 1 implies an additive model, 2 implies a model with up to 2-way interactions, etc. The higher interaction.depth value, the more spread out the importance among predictors. Therefore, the smaller slope of predictor importance figure.

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:

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

```r
library(mice)
library(dplyr)
library(forecast)
library(e1071)
library(caret)
library(RWeka)
library(AppliedPredictiveModeling)
library(pls)
```

```r
data(ChemicalManufacturingProcess)

a <- mice(ChemicalManufacturingProcess, m = 1, method = "pmm", print = F)
C_M_P <- complete(a)

result <- C_M_P %>%
  select(everything()) %>%
  summarize_all(funs(sum(is.na(.))))
data.frame(sort(result, decreasing = TRUE))
CMP_trans <- preProcess(C_M_P, method = c("nzv", "BoxCox", "center",
"scale"))
transformed <- predict(CMP_trans, C_M_P)
```

```
set.seed(88)
trainingRows <- createDataPartition(transformed$Yield, p = .80, list= FALSE)

yield_train <- transformed[trainingRows, 1]
predictor_train <- transformed[trainingRows, -1]

yield_test <- transformed[-trainingRows, 1]
predictor_test <- transformed[-trainingRows, -1]
```

```
ctrl <- trainControl(method = "cv", number = 10)

rpartTune  <- train(x = predictor_train, y = yield_train, method = "rpart2",
tuneLength = 10,trControl = ctrl)
m5Tune <- train(x = predictor_train, y = yield_train, method = "M5",
trControl = ctrl, control = Weka_control(M = 10))
btTune  <- train(x = predictor_train, y = yield_train, method = "treebag",
trControl = ctrl)
rfTune  <- train(x = predictor_train, y = yield_train, method = "rf",
trControl = ctrl, importance=T)
cubistTune <- train(x = predictor_train, y = yield_train,method = "cubist")
```

```
rpart_RMSE <- min(rpartTune$results$RMSE)
m5_RMSE <- min(m5Tune$results$RMSE)
bt_RMSE <- min(btTune$results$RMSE)
rf_RMSE <- min(rfTune$results$RMSE)
cubist_RMSE <- min(cubistTune$results$RMSE)

result <- data.frame(Model = c("Single Trees", "Model Trees", "Bagged Trees",
"Random Forest", "Cubist"), RMSE = c(rpart_RMSE, m5_RMSE, bt_RMSE, rf_RMSE,
cubist_RMSE))
result
```

```
##            Model      RMSE
## 1   Single Trees 0.7463874
## 2    Model Trees 0.6259828
## 3   Bagged Trees 0.6436255
## 4  Random Forest 0.5992231
## 5        Cubist 0.5798986
```

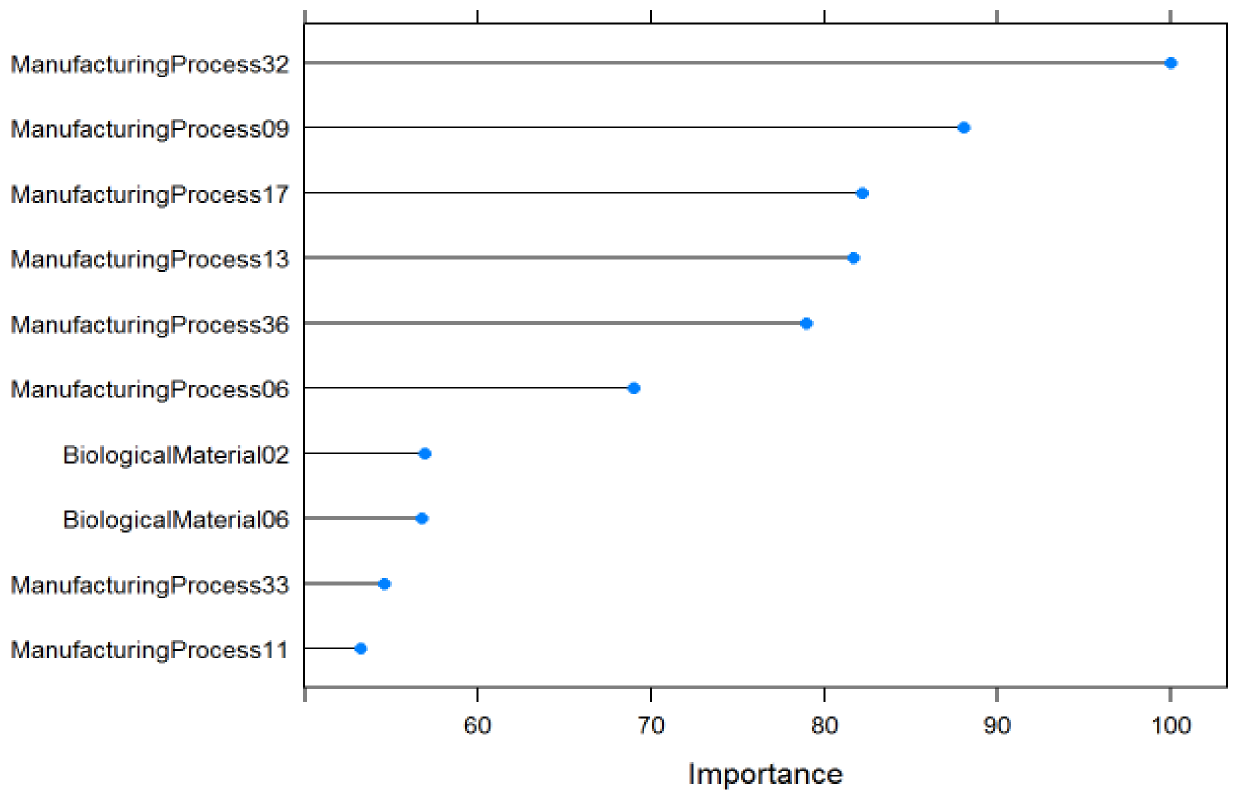
Interpretation: Cubist model gives the optimal resampling and test set performance since it generates lowest RMSE value of `5798986`.

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 nonlinear models?

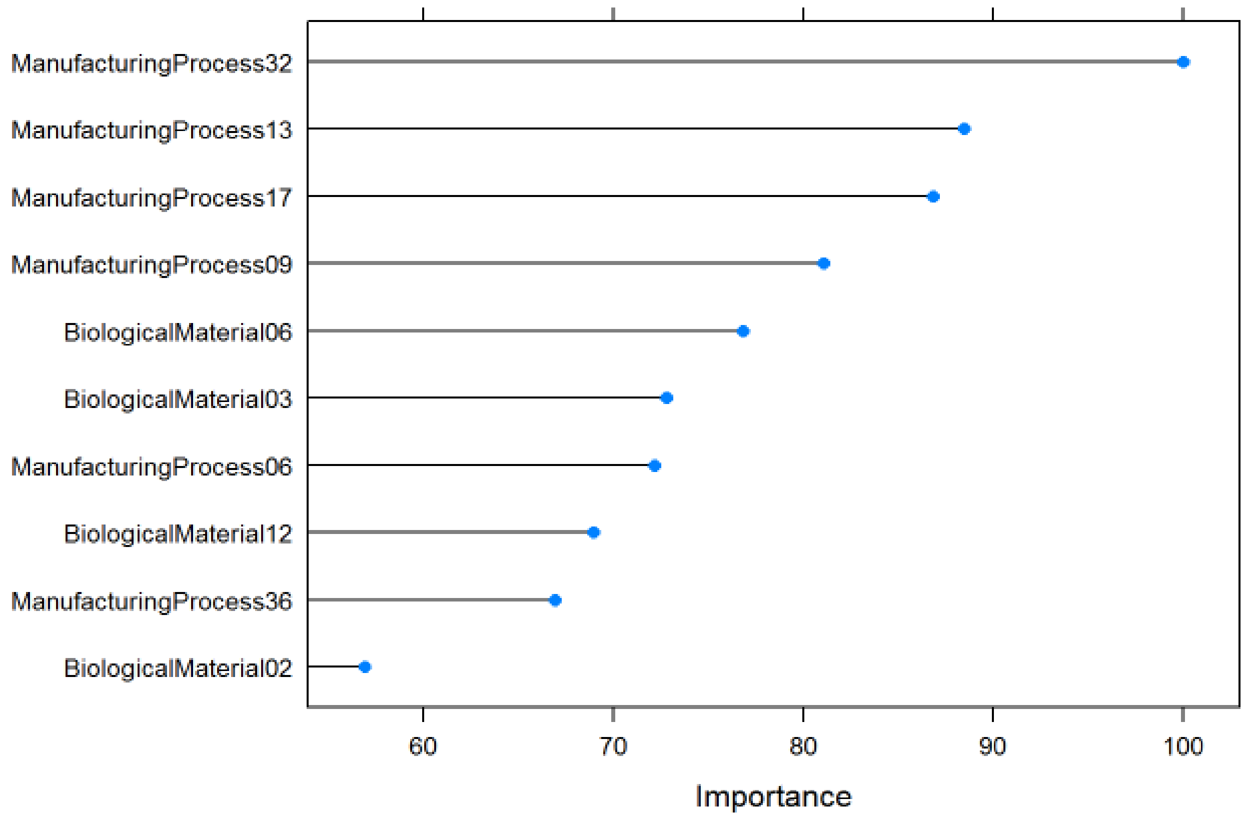
Interpretation: ManufacturingProcess32 is the most important in the optimal tree-based regression model. Manufacturing processes is apparently dominating the list, because they make up 12 out of the top 20 most important predictors. In addition, ManufacturingProcess32 alone includes around 60% overall weight. For linear models, top 5 out of top 10 are all from manufacturing process, on the other hand, nonlinear model gives a little more weight to biological material variables. For tree-based regression model, even more weight is given to biological material variables. However, the slope becomes steeper since big portion of the importance is given to only one variable ManufacturingProcess32.

```
rpartimp3 <- varImp(rpartTune)
m5imp3 <- varImp(m5Tune)
btimp3 <- varImp(btTune)
rfimp3 <- varImp(rfTune)
cunistimp3 <- varImp(cubistTune)

rfimp3
## rf variable importance
##
##   only 20 most important variables shown (out of 56)
##
##                        Overall
## ManufacturingProcess32  100.00
## BiologicalMaterial12     37.09
## ManufacturingProcess17   36.89
## BiologicalMaterial03     36.39
## ManufacturingProcess31   33.75
## ManufacturingProcess13   31.78
## BiologicalMaterial06     31.75
## BiologicalMaterial05     29.73
## BiologicalMaterial11     28.40
## ManufacturingProcess06   28.17
## ManufacturingProcess09   27.58
## ManufacturingProcess11   26.75
## ManufacturingProcess36   24.03
## BiologicalMaterial01     23.19
## ManufacturingProcess18   22.59
## BiologicalMaterial08     22.27
## BiologicalMaterial02     21.95
## ManufacturingProcess25   21.68
## ManufacturingProcess39   21.20
## ManufacturingProcess01   20.81
```

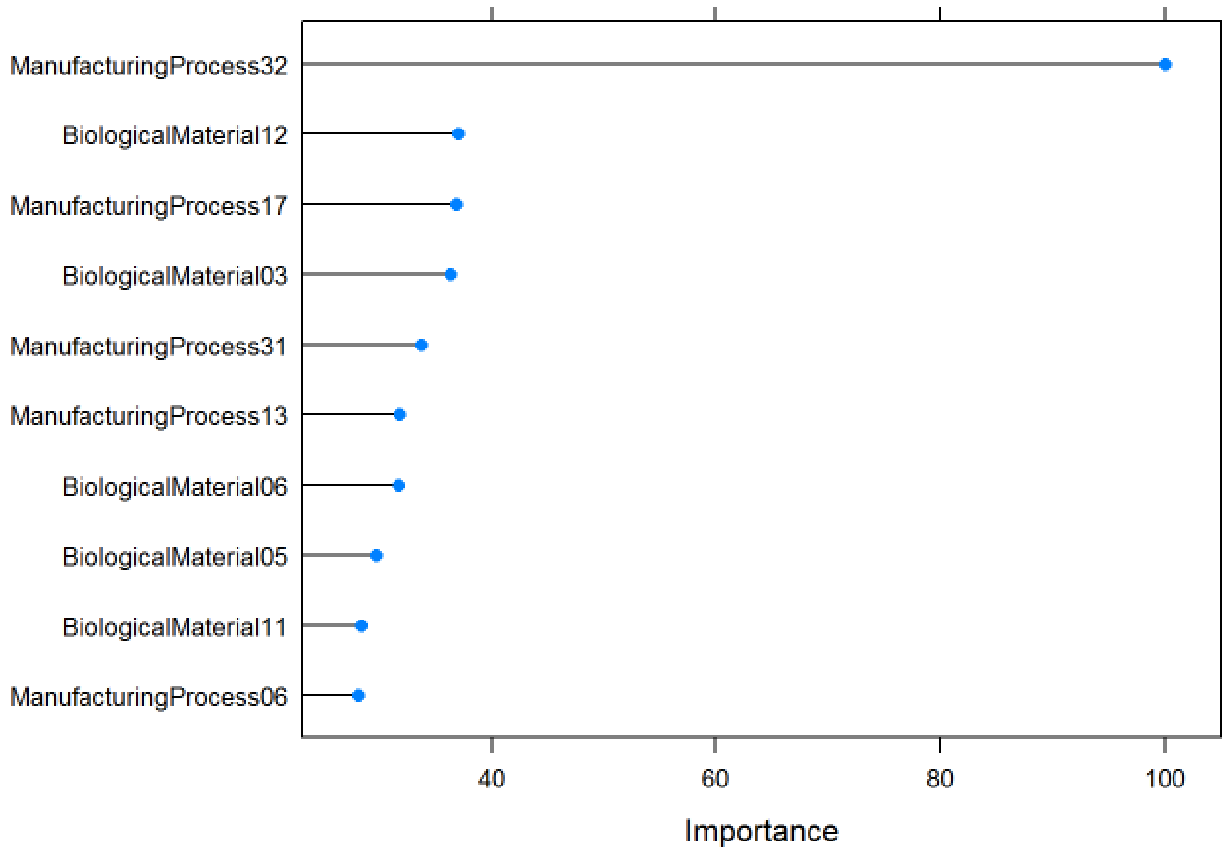
```
set.seed(88)
plsTune <- train(predictor_train, yield_train, method = "pls", tuneLength =
20, trControl = ctrl)
svmRTune <- train(predictor_train, yield_train, method = "svmRadial",
tuneLength = 20, trControl = ctrl)



par(mfrow = c(1, 3))
plot(varImp(plsTune), top = 10)
```



```
plot(varImp(svmRTune), top = 10)
```



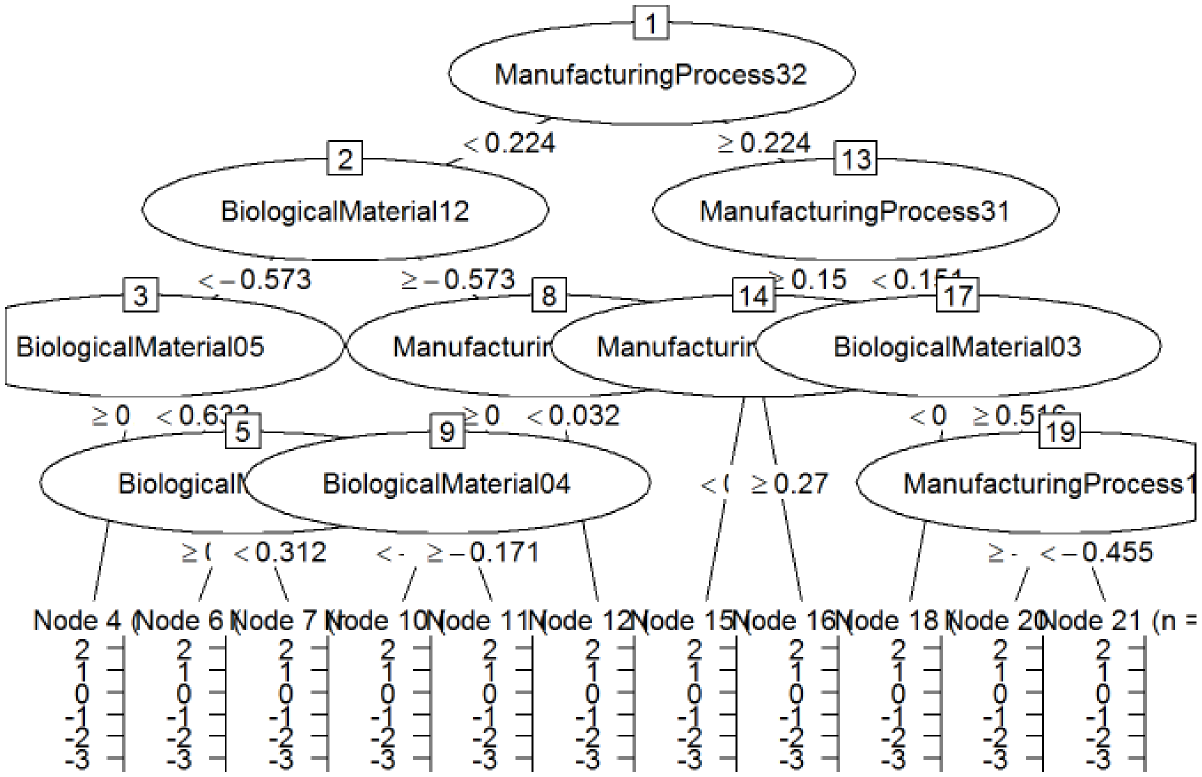
```r
plot(varImp(rfTune), top = 10)
```



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?

Interpretation: The optimal single tree shows ManufacturingProcess32 is the most important predictor, followed by BiologicalMaterial12 and ManufacturingProcess31. The tree also display the cutoff value for each predictor. For example: for ManufacturingProcess32, the cutoff point is 0.224.

```r
library(partykit)
rpartTree <- as.party(rpartTune$finalModel)
plot(rpartTree)
```


1
ManufacturingProcess32
< 0.224
≥ 0.224
2
BiologicalMaterial12
13
ManufacturingProcess31
< −0.573
≥ −0.573
≥ 0.15
< 0.15
3
BiologicalMaterial05
8
14
17
BiologicalMaterial03
≥ 0
< 0.632
5
9
BiologicalMaterial04
≥ 0
< 0.032
< 0
≥ 0.516
19
≥ 0.27
< 0.312
≥ −0.171
< −0.455
Node 4
Node 6
Node 7
Node 10
Node 11
Node 12
Node 15
Node 16
Node 18
Node 20
Node 21 (n =
2
1
0
-1
-2
-3

Homework KJ Chapter 8 Continue:

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

a. A simple regression tree

```
library(caret)
library(AppliedPredictiveModeling)
data("solubility")

sol_Train_X <- subset(solTrainXtrans, select = "NumCarbon")
sol_Train_Y <- solTrainY

sol_Test_X <- subset(solTestX, select = "NumCarbon")
sol_Test_Y <- solTestY
```

```
set.seed(888)
rpartTune  <- train(sol_Train_X, sol_Train_Y, method = "rpart2")
rpartTune
```

```
## CART
##
## 951 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 951, 951, 951, 951, 951, 951, ...
## Resampling results across tuning parameters:
##
##   maxdepth  RMSE      Rsquared   MAE
##   1         1.622600  0.3655507  1.261489
##   2         1.558405  0.4148207  1.212956
##   3         1.497246  0.4599550  1.171599
##
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was maxdepth = 3.
```

b. A random forest model

```
set.seed(888)
rfTune  <- train(sol_Train_X, sol_Train_Y, method = "rf", tuneLength = 1)
rfTune
```

```
## Random Forest
##
## 951 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 951, 951, 951, 951, 951, 951, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.499159  0.4599305  1.171506
##
## Tuning parameter 'mtry' was held constant at a value of 1
```

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

Approaches: RMSE was used to select the optimal model using the smallest value. The final values used for the model were committees = 1 and neighbors = 0.

```
set.seed(888)
grid <- expand.grid(committees = c(1, 10, 50, 100), neighbors = c(0, 1, 5,
9))
```

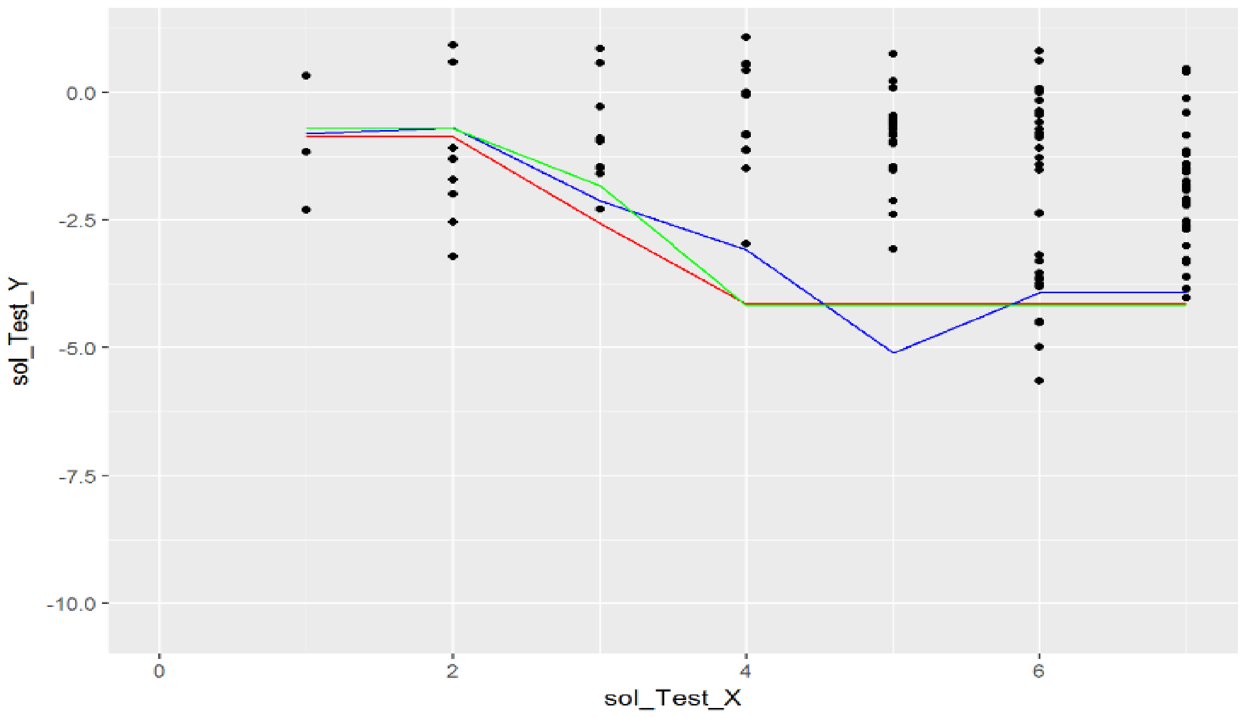
```r
cubistTune <- train(sol_Train_X, sol_Train_Y, method = "cubist",  tuneGrid =
grid)

cubistTune
```

```
## Cubist
##
## 951 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 951, 951, 951, 951, 951, 951, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##     1         0          1.513696  0.4598572  1.159236
##     1         1          1.591658  0.4297735  1.235227
##     1         5          1.590905  0.4301010  1.234767
##     1         9          1.588607  0.4310926  1.233317
##    10         0          1.523531  0.4536249  1.164063
##    10         1          1.592666  0.4292480  1.235701
##    10         5          1.591908  0.4295842  1.235233
##    10         9          1.589577  0.4306198  1.233813
##    50         0          1.519819  0.4558860  1.162366
##    50         1          1.592385  0.4293843  1.235568
##    50         5          1.591627  0.4297223  1.235095
##    50         9          1.589272  0.4307587  1.233646
##   100         0          1.518559  0.4565576  1.162328
##   100         1          1.592505  0.4293628  1.235732
##   100         5          1.591751  0.4296976  1.235265
##   100         9          1.589388  0.4307384  1.233822
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 1 and neighbors = 0.
```

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

Interpretation: From the following figure, all three models generate similar solubility predictions for number of carbons range from 1 to 7. The random forest model has relatively higher variability. This indicates that this model maybe more biased compare to the other ones.

```r
library(ggplot2)


rpartPred <- predict(rpartTune, newdata = sol_Test_X)

rfPred <- predict(rfTune, newdata = sol_Test_X)

cubistPred <- predict(cubistTune, newdata = sol_Test_X)


test <- cbind(sol_Test_X, sol_Test_Y, simple_regression = rpartPred,
random_forest = rfPred, cubist = cubistPred)


ggplot(data = test, aes(x = sol_Test_X, y = sol_Test_Y)) +

  geom_point() +

  xlim(0, 7) +

  geom_line(aes(y = rpartPred), color = "red") +

  geom_line(aes(y = rfPred), color = "blue") +

  geom_line(aes(y = cubistPred), color = "green")
```



8.5. 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?

```r
library(caret)
data(tecator)
set.seed(888)

endpoints1 <- as.data.frame(endpoints)
absorp1 <- as.data.frame(absorp)

trainingRows <- createDataPartition(endpoints1[, 2], p = 0.8, list= FALSE)
endpoints_train <- endpoints1[trainingRows, 2]
endpoints_test <- endpoints1[-trainingRows, 2]

absorp_train <- absorp1[trainingRows, ]
absorp_test <- absorp1[-trainingRows, ]
set.seed(888)

ctrl <- trainControl(method = "cv", preProc = c("center", "scale"))

plsTune2  <- train(x = absorp_train, y = endpoints_train, method = "pls",
trControl = ctrl)
rpartTune2  <- train(x = absorp_train, y = endpoints_train, method =
"rpart2", trControl = ctrl)
rfTune2  <- train(x = absorp_train, y = endpoints_train, method = "rf",
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), n.minobsinnode = 10)
gbmTune2 <- train(x = absorp_train, y = endpoints_train, method = "gbm",
tuneGrid = gbmGrid, verbose = FALSE)

cubistTune2 <- train(x = absorp_train, y = endpoints_train, method =
"cubist")

pls_RMSE2 <- min(plsTune2$results$RMSE)
rpart_RMSE2 <- min(rpartTune2$results$RMSE)
rf_RMSE2 <- min(rfTune2$results$RMSE)
gbm_RMSE2 <- min(gbmTune2$results$RMSE)
cubist_RMSE2 <- min(cubistTune2$results$RMSE)
```

```
result2 <- data.frame(Model = c("pls", "rpart", "rf", "gbm", "cubist"), RMSE
= c(pls_RMSE2, rpart_RMSE2, rf_RMSE2, gbm_RMSE2, cubist_RMSE2))


result2
```

```
##    Model      RMSE
## 1    pls  5.213722
## 2  rpart 10.551262
## 3     rf  7.298284
## 4    gbm  7.421699
## 5 cubist  2.914658
```

Interpretation: Based on RMSE that is generated from each model, we can tell that the Cubist model has the best performance. Among all the models, CART model (Classification and Regression Tree) is one of the worst along with single regression trees model. From some previous exercise, we know the predictor correlation will affect the models. The predictors that are highly correlated to each other will receive same level of importance, while decreasing other predictors' importance down. Therefore, the best practice is to eliminate highly correlated predictors.

8.6. 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?

Interpretation: CART model (Classification and Regression Tree) provides the optimal resampling and test set performance, because it has the lowest RMSE value of `10.82113`.

```
data(permeability)
set.seed(888)


fingerprints1 <- as.data.frame(fingerprints)
permeability1 <- as.data.frame(permeability)


fingerprints1 <- fingerprints1[, -nearZeroVar(fingerprints1)]


trainingRows <- createDataPartition(permeability1$permeability, p = 0.8,
list= FALSE)
fingerprints_train <- fingerprints1[trainingRows,]
```

```r
fingerprints_test <- fingerprints1[-trainingRows,]


permeability_train <- permeability1[trainingRows, ]

permeability_test <- permeability1[-trainingRows, ]

set.seed(888)


ctrl <- trainControl(method = "cv", preProc = c("center", "scale"))


plsTune3  <- train(x = fingerprints_train, y = permeability_train, method =
"pls", trControl = ctrl)

rpartTune3  <- train(x = fingerprints_train, y = permeability_train, method =
"rpart2", trControl = ctrl)

rfTune3  <- train(x = fingerprints_train, y = permeability_train, method =
"rf", 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), n.minobsinnode = 10)

gbmTune3 <- train(x = fingerprints_train, y = permeability_train, method =
"gbm", tuneGrid = gbmGrid, verbose = FALSE)


cubistTune3 <- train(x = fingerprints_train, y = permeability_train, method =
"cubist")

pls_RMSE3 <- min(plsTune3$results$RMSE)

rpart_RMSE3 <- min(rpartTune3$results$RMSE)

rf_RMSE3 <- min(rfTune3$results$RMSE)

gbm_RMSE3 <- min(gbmTune3$results$RMSE)

cubist_RMSE3 <- min(cubistTune3$results$RMSE)


result3 <- data.frame(Model = c("pls", "rpart", "rf", "gbm", "cubist"), RMSE
= c(pls_RMSE3, rpart_RMSE3, rf_RMSE3, gbm_RMSE3, cubist_RMSE3))



result3
```

```
##    Model     RMSE

## 1    pls 11.51701

## 2  rpart 10.82113

## 3     rf 11.01250

## 4    gbm 11.00841
## 5 cubist 11.94154
```

b. Do any of these models outperform the covariance or non-covariance based regression models you have previously developed for these data? What criteria did you use to compare models' performance?

Interpreation: None of these models outperforms the covariance or non-covariance based regression models than SVM radial model. I am using RMSE to compare models' performance.SVM radial model obtains RMSE value of 9.85278, which is the lowest among all models.

```r
set.seed(888)

ctrl <- trainControl(method = "cv", preProc = c("center", "scale",
"knnImpute"))


knnTune3 <- train(x = fingerprints_train, y = permeability_train, method =
"knn", tuneLength = 10)


marsGrid <- expand.grid(degree = 1:2, nprune = 1:20)

marsTune3 <- train(x = fingerprints_train, y = permeability_train, method =
"earth", tuneGrid = marsGrid, trControl = ctrl)

svmRTune3 <- train(x = fingerprints_train, y = permeability_train, method =
"svmRadial", tuneLength = 20, trControl = ctrl)

knn_RMSE3 <- min(knnTune3$results$RMSE)

mars_RMSE3 <- min(marsTune3$results$RMSE)

svmR_RMSE3 <- min(svmRTune3$results$RMSE)


result4 <- data.frame(Model = c("knn", "mars", "svmR"), RMSE = c(knn_RMSE3,
mars_RMSE3, svmR_RMSE3))


result4
```

```
##   Model     RMSE
## 1   knn 12.75464
## 2  mars 10.80341
## 3  svmR  9.85278
```

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

I would recommend SVM model with svmRadial method, because it generate lowest RMSE value.

Market Basket Analysis

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.

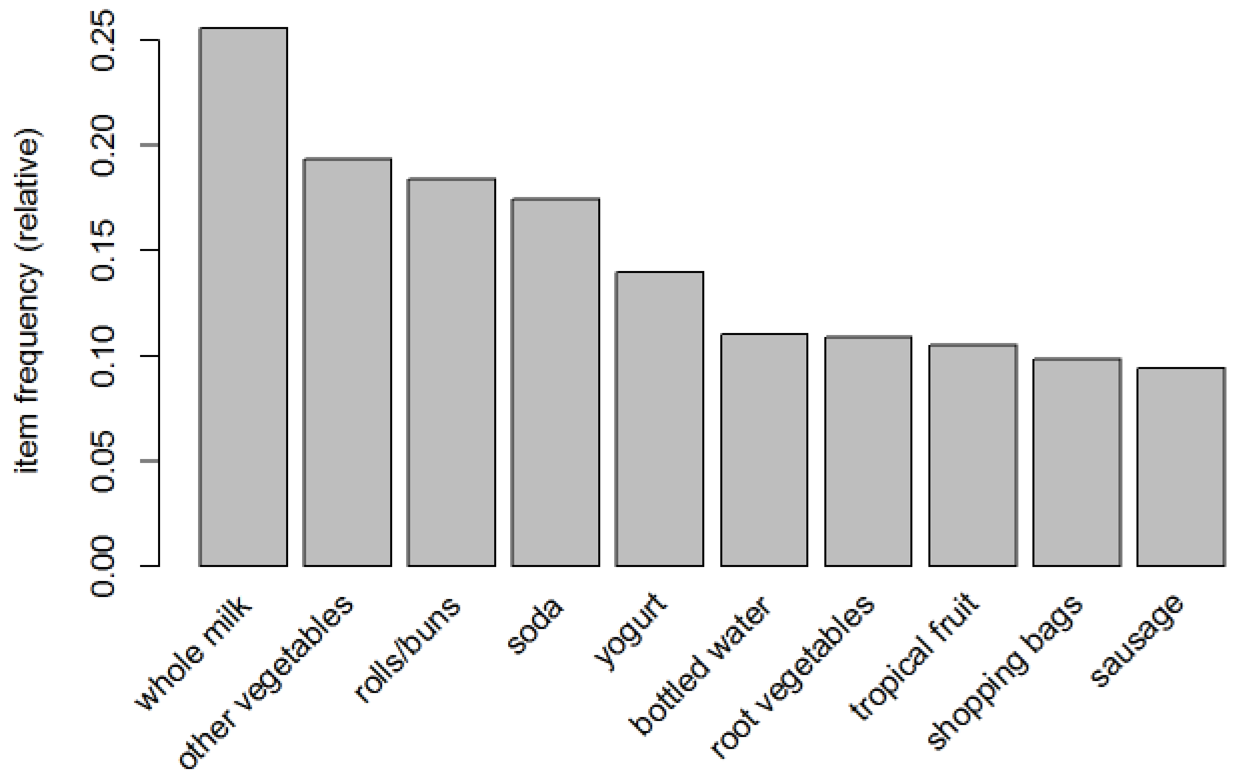
Approaches: For this assignment, I will need the R package called arules. This package can be used for mining association rules and frequent itemsets. From the following summary statistics, we know this dataset contains 9835 transactions and 169 items.

```
library(arules)

raw_data <-
read.transactions("https://raw.githubusercontent.com/blin261/624/master/Groce
ryDataSet.csv", format = "basket", sep = ",")


summary(raw_data)
```

```
## 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
```

Approaches: First of all, I would like to investigate the frequencies of each item and filter out the top 10 items using itemFrequencyPlot method.

```
itemFrequencyPlot(raw_data, topN = 10)
```



Association Rules: Support is an indication of how frequently the itemset appears in the dataset. Confidence is an indication of how often the rule has been found to be true. Lift is the ratio of the observed support to that expected

The following result shows the summary statistics of the apriori algorithms. There are total 410 rules with the length distributed from 3 to 6. The distribution for support, confidence, and lift are also shown as follows.

```
rules <- apriori(raw_data, parameter = list(supp = 0.001, confidence = 0.8,
minlen = 2), control = list(verbose = FALSE))

summary(rules)
```

```
## set of 410 rules
```

```
##
## rule length distribution (lhs + rhs):sizes
##   3   4   5   6
##  29 229 140  12
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   4.000   4.000   4.329   5.000   6.000
##
## summary of quality measures:
##     support           confidence          lift            count
##  Min.   :0.001017   Min.   :0.8000   Min.   : 3.131   Min.   :10.00
##  1st Qu.:0.001017   1st Qu.:0.8333   1st Qu.: 3.312   1st Qu.:10.00
##  Median :0.001220   Median :0.8462   Median : 3.588   Median :12.00
##  Mean   :0.001247   Mean   :0.8663   Mean   : 3.951   Mean   :12.27
##  3rd Qu.:0.001322   3rd Qu.:0.9091   3rd Qu.: 4.341   3rd Qu.:13.00
##  Max.   :0.003152   Max.   :1.0000   Max.   :11.235   Max.   :31.00
##
## mining info:
##      data ntransactions support confidence
##  raw_data          9835   0.001        0.8
```

The following code will order all the rules by descending order based on lift. Then I picked up the top 10 rules and investigated further.

```
inspect(head(rules, 10, by = "lift"))
##      lhs                            rhs                     support confidence
lift count
## [1]  {liquor,
##       red/blush wine}         => {bottled beer}    0.001931876  0.9047619
11.235269    19
## [2]  {citrus fruit,
##       fruit/vegetable juice,
##       other vegetables,
##       soda}                  => {root vegetables} 0.001016777  0.9090909
8.340400    10
## [3]  {oil,
##       other vegetables,
##       tropical fruit,
##       whole milk,
##       yogurt}                => {root vegetables} 0.001016777  0.9090909
8.340400    10
## [4]  {citrus fruit,
##       fruit/vegetable juice,
##       grapes}                => {tropical fruit}  0.001118454  0.8461538
8.063879    11
## [5]  {other vegetables,
##       rice,
##       whole milk,
##       yogurt}                => {root vegetables} 0.001321810  0.8666667
7.951182    13
## [6]  {oil,
##       other vegetables,
##       tropical fruit,
##       whole milk}            => {root vegetables} 0.001321810  0.8666667
7.951182    13
## [7]  {ham,
##       other vegetables,
##       pip fruit,
##       yogurt}                => {tropical fruit}  0.001016777  0.8333333
7.941699    10
## [8]  {beef,
##       citrus fruit,
##       other vegetables,
##       tropical fruit}        => {root vegetables} 0.001016777  0.8333333
7.645367    10
## [9]  {butter,
##       cream cheese,
##       root vegetables}       => {yogurt}          0.001016777  0.9090909
6.516698    10
## [10] {butter,
##       sliced cheese,
##       tropical fruit,
##       whole milk}            => {yogurt}          0.001016777  0.9090909
6.516698    10
```

Interpretation: The rule with the highest lift is for a purchase of bottled beer after purchase of liquor,red/blush wine. The rule has the lift value at 11.235. This pattern appears 19 times in the datasets.