Data624 HW2

Samuel Kigamba, Lin Li, Patrick Maloney, Daniel Moscoe, David Moste

7/16/2021

# KJ 6.3

#### Question

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:

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

#### Code

library(AppliedPredictiveModeling)  
data(ChemicalManufacturingProcess)

#### Response

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.

#### Question

1. 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).

#### Code

# Get total number of missing values  
sum(is.na(ChemicalManufacturingProcess))

## [1] 106

# Impute data with "knnImpute" method  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(RANN)  
  
Chem\_impute <- preProcess(ChemicalManufacturingProcess, method = "knnImpute")  
impute\_Result <- predict(Chem\_impute, ChemicalManufacturingProcess)  
sum(is.na(impute\_Result))

## [1] 0

#### Question

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?

#### Code

library(caret)  
  
# Box–Cox transform and scale   
trans <- preProcess(impute\_Result, method = c("BoxCox", "scale"))

library(caret)  
  
transformed <- predict(trans, impute\_Result)

library(caret)  
  
# Split data into 75 % training and 25 % testing sets  
set.seed(1)  
n <- nrow(transformed)  
trainIndex <- sample(1:n, size = round(0.75\*n), replace=FALSE)  
train <- transformed[trainIndex ,]  
test <- transformed[-trainIndex ,]  
x <- train[, -1]  
y <- train[, 1]  
set.seed(100)  
plsTune <- train(x, y,   
 method = "pls",  
 tuneLength = 20,  
 trControl = trainControl(method = "cv", number = 10))  
plsTune

## Partial Least Squares   
##   
## 132 samples  
## 57 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 119, 119, 118, 120, 119, 118, ...   
## Resampling results across tuning parameters:  
##   
## ncomp RMSE Rsquared MAE   
## 1 0.7570309 0.4695266 0.6228282  
## 2 0.6945795 0.5667242 0.5513665  
## 3 0.6725321 0.6121573 0.5477136  
## 4 0.6701932 0.6229922 0.5476533  
## 5 0.6870380 0.6073507 0.5715733  
## 6 0.6961185 0.5937035 0.5778664  
## 7 0.7065944 0.5851510 0.5878432  
## 8 0.7153011 0.5861943 0.5948410  
## 9 0.7338339 0.5728594 0.6022937  
## 10 0.7632407 0.5612240 0.6244762  
## 11 0.7857860 0.5511652 0.6343056  
## 12 0.8050340 0.5442486 0.6445692  
## 13 0.8161405 0.5450546 0.6518595  
## 14 0.8353229 0.5438029 0.6608138  
## 15 0.8841104 0.5290770 0.6830433  
## 16 0.9232712 0.5300995 0.6992455  
## 17 0.9494331 0.5224930 0.7095627  
## 18 0.9731843 0.5186949 0.7198537  
## 19 1.0192248 0.4994294 0.7406936  
## 20 1.0596215 0.4857855 0.7562482  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was ncomp = 4.

library(caret)  
  
plsTunePred <- predict(plsTune, test)

#### Question

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 metricon the training set?

#### Code

library(pls)

##   
## Attaching package: 'pls'

## The following object is masked from 'package:caret':  
##   
## R2

## The following object is masked from 'package:stats':  
##   
## loadings

plsFit <- plsr(Yield ~., data = train)  
plsPred <- predict(plsFit, test)

#### Question

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

#### Code

library(mlbench)  
  
# Rank variables by importance  
importance <- varImp(plsTune, scale = FALSE)  
print(importance)

## pls variable importance  
##   
## only 20 most important variables shown (out of 57)  
##   
## Overall  
## ManufacturingProcess32 0.06477  
## ManufacturingProcess13 0.06340  
## ManufacturingProcess09 0.06222  
## ManufacturingProcess17 0.05909  
## ManufacturingProcess36 0.05459  
## ManufacturingProcess12 0.04499  
## BiologicalMaterial03 0.04470  
## BiologicalMaterial02 0.04385  
## BiologicalMaterial06 0.04279  
## ManufacturingProcess06 0.04112  
## BiologicalMaterial12 0.04077  
## BiologicalMaterial11 0.04058  
## ManufacturingProcess33 0.04051  
## BiologicalMaterial08 0.04032  
## BiologicalMaterial01 0.03725  
## BiologicalMaterial04 0.03442  
## ManufacturingProcess11 0.03416  
## ManufacturingProcess28 0.03346  
## ManufacturingProcess04 0.02627  
## BiologicalMaterial09 0.02508

#### Response

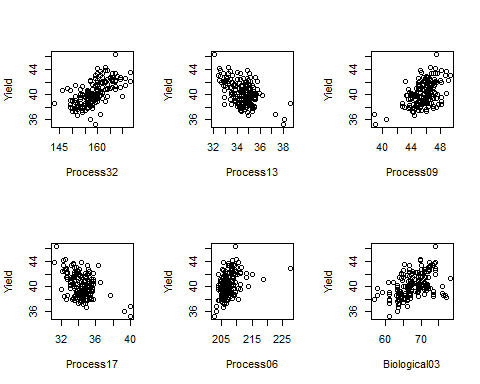
The first six most important predictors are dominated by manufacturing processes. The number of biological and process predictors are similar for the top 20 most important predictors.

#### Question

1. 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?

#### Code

par(mfrow=c(2,3))  
plot(ChemicalManufacturingProcess$ManufacturingProcess32, ChemicalManufacturingProcess$Yield, xlab = "Process32", ylab = "Yield")  
plot(ChemicalManufacturingProcess$ManufacturingProcess13, ChemicalManufacturingProcess$Yield, xlab = "Process13", ylab = "Yield")  
plot(ChemicalManufacturingProcess$ManufacturingProcess09, ChemicalManufacturingProcess$Yield, xlab = "Process09", ylab = "Yield")  
plot(ChemicalManufacturingProcess$ManufacturingProcess17, ChemicalManufacturingProcess$Yield, xlab = "Process17", ylab = "Yield")  
plot(ChemicalManufacturingProcess$ManufacturingProcess06, ChemicalManufacturingProcess$Yield,xlab = "Process06", ylab = "Yield")  
plot(ChemicalManufacturingProcess$BiologicalMaterial03, ChemicalManufacturingProcess$Yield,xlab = "Biological03", ylab = "Yield")



#### Response

The plots suggest that the most important variables are either positively or negatively correlated to Yield. Since the manufacturing predictors are the most important, removing/modifying the ones that have negative correlations with Yield can improve the yield production.

# KJ 7.2

#### Question

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 = 10 sin(πx1x2) + 20(x3 − 0.5)2 + 10x4 + 5x5 + N(0, σ2) 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).

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

#### Code

library(caret)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(tidyr)  
library(corrplot)

## corrplot 0.89 loaded

##   
## Attaching package: 'corrplot'

## The following object is masked from 'package:pls':  
##   
## corrplot

library(AppliedPredictiveModeling)  
library(e1071)  
library(kernlab)

##   
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':  
##   
## alpha

library(earth)

## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos

library(dplyr)  
library(mlbench)  
library(RANN)  
  
set.seed(200)  
trainingData <- mlbench.friedman1(200, sd = 1) ##We convert the 'x' data from a matrix to a data frame > ##One reason is that this will give the columns names.   
trainingData$x <- data.frame(trainingData$x) ##Look at the data using > featurePlot(trainingData$x, trainingData$y) > ##or other methods. > > ##This creates a list with a vector 'y' and a matrix > ##of predictors 'x'. Also simulate a large test set to > ##estimate the true error rate with good precision: >   
testData <- mlbench.friedman1(5000, sd = 1)   
testData$x <- data.frame(testData$x)  
  
# KNN  
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 3.466085 0.5121775 2.816838  
## 7 3.349428 0.5452823 2.727410  
## 9 3.264276 0.5785990 2.660026  
## 11 3.214216 0.6024244 2.603767  
## 13 3.196510 0.6176570 2.591935  
## 15 3.184173 0.6305506 2.577482  
## 17 3.183130 0.6425367 2.567787  
## 19 3.198752 0.6483184 2.592683  
## 21 3.188993 0.6611428 2.588787  
## 23 3.200458 0.6638353 2.604529  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 17.

knnPred <- predict(knnModel, newdata = testData$x) ##The function 'postResample' can be used to get the test set   
##perforamnce values   
postResample(pred = knnPred, obs = testData$y)

## RMSE Rsquared MAE   
## 3.2040595 0.6819919 2.5683461

# MARS  
marsFit <- earth(trainingData$x, trainingData$y)  
marsFit

## Selected 12 of 18 terms, and 6 of 10 predictors  
## Termination condition: Reached nk 21  
## Importance: X1, X4, X2, X5, X3, X6, X7-unused, X8-unused, X9-unused, ...  
## Number of terms at each degree of interaction: 1 11 (additive model)  
## GCV 2.540556 RSS 397.9654 GRSq 0.8968524 RSq 0.9183982

summary(marsFit)

## Call: earth(x=trainingData$x, y=trainingData$y)  
##   
## coefficients  
## (Intercept) 18.451984  
## h(0.621722-X1) -11.074396  
## h(0.601063-X2) -10.744225  
## h(X3-0.281766) 20.607853  
## h(0.447442-X3) 17.880232  
## h(X3-0.447442) -23.282007  
## h(X3-0.636458) 15.150350  
## h(0.734892-X4) -10.027487  
## h(X4-0.734892) 9.092045  
## h(0.850094-X5) -4.723407  
## h(X5-0.850094) 10.832932  
## h(X6-0.361791) -1.956821  
##   
## Selected 12 of 18 terms, and 6 of 10 predictors  
## Termination condition: Reached nk 21  
## Importance: X1, X4, X2, X5, X3, X6, X7-unused, X8-unused, X9-unused, ...  
## Number of terms at each degree of interaction: 1 11 (additive model)  
## GCV 2.540556 RSS 397.9654 GRSq 0.8968524 RSq 0.9183982

marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)   
marsFittune <- train(trainingData$x, trainingData$y, method = "earth", tuneGrid = marsGrid, trControl = trainControl(method = "cv"))   
summary(marsFittune)

## Call: earth(x=data.frame[200,10], y=c(18.46,16.1,17...), keepxy=TRUE, degree=2,  
## nprune=13)  
##   
## coefficients  
## (Intercept) 22.050690  
## h(0.621722-X1) -15.001651  
## h(X1-0.621722) 10.878737  
## h(0.601063-X2) -18.830135  
## h(0.447442-X3) 9.940077  
## h(X3-0.606015) 12.999390  
## h(0.734892-X4) -9.877554  
## h(X4-0.734892) 10.414930  
## h(0.850094-X5) -5.604897  
## h(X1-0.621722) \* h(X2-0.295997) -43.245766  
## h(0.649253-X1) \* h(0.601063-X2) 26.218297  
##   
## Selected 11 of 18 terms, and 5 of 10 predictors (nprune=13)  
## Termination condition: Reached nk 21  
## Importance: X1, X4, X2, X5, X3, X6-unused, X7-unused, X8-unused, X9-unused, ...  
## Number of terms at each degree of interaction: 1 8 2  
## GCV 1.747495 RSS 264.5358 GRSq 0.929051 RSq 0.9457576

head(predict(marsFittune, testData$x))

## y  
## [1,] 18.539042  
## [2,] 21.224619  
## [3,] 11.994784  
## [4,] 8.087937  
## [5,] 11.415761  
## [6,] 12.293669

varImp(marsFittune)

## earth variable importance  
##   
## Overall  
## X1 100.00  
## X4 75.33  
## X2 48.88  
## X5 15.63  
## X3 0.00

#### Response

The optimal MARS model minimized the RMSE when the nprune = 13 and the degree = 2. MARS selected x1 as the most important predictor and x4 as second. The model used 152 training set data points as support vectors. The MARS Model fit best with a high R2 of ~.95

# KJ 7.5

#### Question

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.

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

#### Code

library(AppliedPredictiveModeling)  
library(caret)  
  
data(ChemicalManufacturingProcess)  
set.seed(56)  
knnmodel2 <- preProcess(ChemicalManufacturingProcess, "knnImpute")  
df <- predict(knnmodel2, ChemicalManufacturingProcess)  
df <- df %>%  
 select\_at(vars(-one\_of(nearZeroVar(., names = TRUE))))  
in\_train <- createDataPartition(df$Yield, times = 1, p = 0.8, list = FALSE)  
train\_df <- df[in\_train, ]  
test\_df <- df[-in\_train, ]  
  
# KNN  
knn\_model <- train(  
 Yield ~ ., data = train\_df, method = "knn",  
 center = TRUE,  
 scale = TRUE,  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 25  
)  
knn\_model

## k-Nearest Neighbors   
##   
## 144 samples  
## 56 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 130, 129, 131, 130, 130, 130, ...   
## Resampling results across tuning parameters:  
##   
## k RMSE Rsquared MAE   
## 5 0.6665611 0.6064388 0.5302668  
## 7 0.7087263 0.5402774 0.5678798  
## 9 0.7130702 0.5331043 0.5820999  
## 11 0.7161287 0.5489623 0.5796126  
## 13 0.7126469 0.5579189 0.5727752  
## 15 0.7244959 0.5500764 0.5840181  
## 17 0.7354577 0.5430152 0.5902397  
## 19 0.7423521 0.5386067 0.5961459  
## 21 0.7615809 0.5095456 0.6142849  
## 23 0.7699950 0.4994279 0.6197973  
## 25 0.7795341 0.4884387 0.6250342  
## 27 0.7825606 0.4899842 0.6296604  
## 29 0.7860966 0.4941641 0.6328056  
## 31 0.7939608 0.4865922 0.6400718  
## 33 0.8040618 0.4650490 0.6460752  
## 35 0.8070461 0.4670781 0.6454113  
## 37 0.8127633 0.4634604 0.6505921  
## 39 0.8176406 0.4642579 0.6554725  
## 41 0.8260090 0.4508073 0.6624975  
## 43 0.8236464 0.4664046 0.6614813  
## 45 0.8290918 0.4530441 0.6652781  
## 47 0.8344262 0.4490136 0.6709419  
## 49 0.8424615 0.4305877 0.6796501  
## 51 0.8481423 0.4220161 0.6835723  
## 53 0.8523472 0.4237087 0.6865206  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 5.

knn\_predictions <- predict(knn\_model, test\_df)  
results <- data.frame(t(postResample(pred = knn\_predictions, obs = test\_df$Yield))) %>%  
 mutate("Model"= "KNN")   
results

## RMSE Rsquared MAE Model  
## 1 0.7242108 0.3334588 0.5853915 KNN

# MARS  
MARS\_grid <- expand.grid(.degree = 1:2, .nprune = 2:38)  
MARS\_model <- train(  
 Yield ~ ., data = train\_df, method = "earth",  
 tuneGrid = MARS\_grid,  
 # If the following lines are uncommented, it throws an error  
 #center = TRUE,  
 #scale = TRUE,  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 25  
)  
MARS\_model

## Multivariate Adaptive Regression Spline   
##   
## 144 samples  
## 56 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 129, 128, 129, 131, 129, 130, ...   
## Resampling results across tuning parameters:  
##   
## degree nprune RMSE Rsquared MAE   
## 1 2 0.7840293 0.4910640 0.6195486  
## 1 3 0.6440776 0.6478745 0.5256738  
## 1 4 0.6019981 0.6842362 0.4905541  
## 1 5 0.6204087 0.6604728 0.5057834  
## 1 6 0.6206660 0.6540201 0.5125711  
## 1 7 0.6270212 0.6627605 0.5226148  
## 1 8 0.6666876 0.6214691 0.5517621  
## 1 9 0.6746168 0.6287750 0.5515463  
## 1 10 0.6696009 0.6301541 0.5543528  
## 1 11 0.6762604 0.6309171 0.5575865  
## 1 12 0.6679343 0.6390047 0.5542393  
## 1 13 0.6831036 0.6274958 0.5619842  
## 1 14 0.6800274 0.6242549 0.5611026  
## 1 15 0.6734382 0.6238292 0.5582140  
## 1 16 0.6734382 0.6238292 0.5582140  
## 1 17 0.6734382 0.6238292 0.5582140  
## 1 18 0.6734382 0.6238292 0.5582140  
## 1 19 0.6734382 0.6238292 0.5582140  
## 1 20 0.6734382 0.6238292 0.5582140  
## 1 21 0.6734382 0.6238292 0.5582140  
## 1 22 0.6734382 0.6238292 0.5582140  
## 1 23 0.6734382 0.6238292 0.5582140  
## 1 24 0.6734382 0.6238292 0.5582140  
## 1 25 0.6734382 0.6238292 0.5582140  
## 1 26 0.6734382 0.6238292 0.5582140  
## 1 27 0.6734382 0.6238292 0.5582140  
## 1 28 0.6734382 0.6238292 0.5582140  
## 1 29 0.6734382 0.6238292 0.5582140  
## 1 30 0.6734382 0.6238292 0.5582140  
## 1 31 0.6734382 0.6238292 0.5582140  
## 1 32 0.6734382 0.6238292 0.5582140  
## 1 33 0.6734382 0.6238292 0.5582140  
## 1 34 0.6734382 0.6238292 0.5582140  
## 1 35 0.6734382 0.6238292 0.5582140  
## 1 36 0.6734382 0.6238292 0.5582140  
## 1 37 0.6734382 0.6238292 0.5582140  
## 1 38 0.6734382 0.6238292 0.5582140  
## 2 2 0.7840293 0.4910640 0.6195486  
## 2 3 0.6747248 0.6159095 0.5529334  
## 2 4 0.6464768 0.6255431 0.5352276  
## 2 5 0.6589820 0.6081124 0.5463171  
## 2 6 0.6612353 0.5993386 0.5518179  
## 2 7 0.6403966 0.6410935 0.5268027  
## 2 8 0.6446978 0.6364260 0.5259487  
## 2 9 0.6508266 0.6284406 0.5305388  
## 2 10 0.6886366 0.5890093 0.5651645  
## 2 11 0.6714691 0.6072878 0.5487666  
## 2 12 1.6321948 0.4793000 0.8506255  
## 2 13 1.6504889 0.4529011 0.8573734  
## 2 14 1.6296744 0.4956339 0.8489240  
## 2 15 1.6334893 0.4981260 0.8590366  
## 2 16 1.6569222 0.5005990 0.8641363  
## 2 17 1.7206211 0.4897833 0.8817245  
## 2 18 1.7243336 0.4817971 0.8953906  
## 2 19 1.7415336 0.4843562 0.9009879  
## 2 20 1.8393685 0.4841982 0.9351759  
## 2 21 1.9474175 0.4724952 0.9770869  
## 2 22 1.9436269 0.4724638 0.9732645  
## 2 23 1.9398849 0.4788261 0.9731627  
## 2 24 1.9342924 0.4799388 0.9703221  
## 2 25 1.9327462 0.4780241 0.9671171  
## 2 26 1.9327462 0.4780241 0.9671171  
## 2 27 1.9275739 0.4749331 0.9587218  
## 2 28 1.9275168 0.4747577 0.9583479  
## 2 29 1.9328082 0.4691185 0.9635021  
## 2 30 1.9327121 0.4689069 0.9624974  
## 2 31 1.9327121 0.4689069 0.9624974  
## 2 32 1.9327121 0.4689069 0.9624974  
## 2 33 1.9327121 0.4689069 0.9624974  
## 2 34 1.9327121 0.4689069 0.9624974  
## 2 35 1.9327121 0.4689069 0.9624974  
## 2 36 1.9327121 0.4689069 0.9624974  
## 2 37 1.9327121 0.4689069 0.9624974  
## 2 38 1.9327121 0.4689069 0.9624974  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were nprune = 4 and degree = 1.

head(predict(MARS\_model, test\_df))

## y  
## [1,] 0.39300400  
## [2,] 1.30236945  
## [3,] 0.09523008  
## [4,] 0.75324247  
## [5,] 0.51308349  
## [6,] -0.39093407

# SVM  
SVM\_model <- train(  
 Yield ~ ., data = train\_df, method = "svmRadial",  
 center = TRUE,  
 scale = TRUE,  
 trControl = trainControl(method = "cv"),  
 tuneLength = 25  
)  
SVM\_model

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 144 samples  
## 56 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 130, 131, 128, 129, 129, 130, ...   
## Resampling results across tuning parameters:  
##   
## C RMSE Rsquared MAE   
## 0.25 0.7440003 0.5285639 0.5993020  
## 0.50 0.6778165 0.5910408 0.5410692  
## 1.00 0.6403732 0.6261244 0.5013226  
## 2.00 0.6121304 0.6557169 0.4788648  
## 4.00 0.6067117 0.6649793 0.4746695  
## 8.00 0.6109981 0.6578050 0.4847679  
## 16.00 0.6105438 0.6566254 0.4843467  
## 32.00 0.6105438 0.6566254 0.4843467  
## 64.00 0.6105438 0.6566254 0.4843467  
## 128.00 0.6105438 0.6566254 0.4843467  
## 256.00 0.6105438 0.6566254 0.4843467  
## 512.00 0.6105438 0.6566254 0.4843467  
## 1024.00 0.6105438 0.6566254 0.4843467  
## 2048.00 0.6105438 0.6566254 0.4843467  
## 4096.00 0.6105438 0.6566254 0.4843467  
## 8192.00 0.6105438 0.6566254 0.4843467  
## 16384.00 0.6105438 0.6566254 0.4843467  
## 32768.00 0.6105438 0.6566254 0.4843467  
## 65536.00 0.6105438 0.6566254 0.4843467  
## 131072.00 0.6105438 0.6566254 0.4843467  
## 262144.00 0.6105438 0.6566254 0.4843467  
## 524288.00 0.6105438 0.6566254 0.4843467  
## 1048576.00 0.6105438 0.6566254 0.4843467  
## 2097152.00 0.6105438 0.6566254 0.4843467  
## 4194304.00 0.6105438 0.6566254 0.4843467  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.01364992  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were sigma = 0.01364992 and C = 4.

SVM\_predictions <- predict(SVM\_model, test\_df)  
results <- data.frame(t(postResample(pred = SVM\_predictions, obs = test\_df$Yield))) %>%  
 mutate("Model"= "SVM")

#### Response

The SVM model was the best model according to the

#### Question

1. 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?

#### Code

library(caret)  
  
varImp(SVM\_model, 10)

## loess r-squared variable importance  
##   
## only 20 most important variables shown (out of 56)  
##   
## Overall  
## ManufacturingProcess13 100.00  
## ManufacturingProcess32 97.81  
## BiologicalMaterial06 86.14  
## ManufacturingProcess17 82.38  
## BiologicalMaterial03 77.56  
## BiologicalMaterial12 77.23  
## ManufacturingProcess09 74.35  
## ManufacturingProcess36 73.84  
## BiologicalMaterial02 63.58  
## ManufacturingProcess31 59.17  
## ManufacturingProcess06 59.09  
## BiologicalMaterial11 52.43  
## ManufacturingProcess29 51.35  
## ManufacturingProcess12 49.40  
## ManufacturingProcess11 49.21  
## ManufacturingProcess02 48.41  
## BiologicalMaterial08 46.71  
## BiologicalMaterial04 45.78  
## ManufacturingProcess33 45.16  
## BiologicalMaterial09 40.05

#### Response

The processing (manufacturing) variables are most important with mfgprocess13 as most important.

#### Question

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?

#### Code

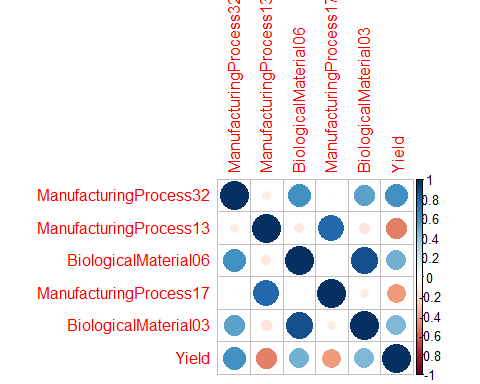
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble 3.1.2 v stringr 1.4.0  
## v readr 1.4.0 v forcats 0.5.1  
## v purrr 0.3.4

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x kernlab::alpha() masks ggplot2::alpha()  
## x purrr::cross() masks kernlab::cross()  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## x purrr::lift() masks caret::lift()

df %>% select(c('ManufacturingProcess32','ManufacturingProcess13','BiologicalMaterial06','ManufacturingProcess17','BiologicalMaterial03','Yield')) %>% cor() %>% corrplot(method = 'circle')



#### Response

Mfgprocess32 shows to have a high positive correlation, while Mfg Process 13 has a strong negative correlation. Biological Material 03 and 06 have positive correlations.

# KJ 8.1

#### Question

Recreate the simulated data from Exercise 7.2:

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

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

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':  
##   
## combine

## The following object is masked from 'package:ggplot2':  
##   
## margin

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

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

#### Code

rfImp1

## Overall  
## V1 8.732235404  
## V2 6.415369387  
## V3 0.763591825  
## V4 7.615118809  
## V5 2.023524577  
## V6 0.165111172  
## V7 -0.005961659  
## V8 -0.166362581  
## V9 -0.095292651  
## V10 -0.074944788

#### Response

The code above shows the overall variable importance scores for the model. Variables V6-V10 have very low scores relative to the informative variables, V1-V5. The random forest model did not significantly use the uninformative predictors.

#### Question

1. Now add an additional predictor that is highly correlated with one of the informative predictors…. Fit another random forest model to these data. Did the importance score for V1 change?

#### Code

library(randomForest)  
  
simulated2 <- mutate(simulated, "duplicate1" = simulated$V1 + rnorm(200) \* 0.1)  
model2 <- randomForest(y ~ ., data = simulated2,  
 importance = TRUE,  
 ntree = 1000)  
rfImp2 <- varImp(model2, scale = FALSE)  
rfImp2

## Overall  
## V1 5.69119973  
## V2 6.06896061  
## V3 0.62970218  
## V4 7.04752238  
## V5 1.87238438  
## V6 0.13569065  
## V7 -0.01345645  
## V8 -0.04370565  
## V9 0.00840438  
## V10 0.02894814  
## duplicate1 4.28331581

#### Response

The importance score for V1 is reduced when another highly correlated predictor variable is added to the model. This example serves as a warning about interpreting results of random forests that contain correlated variables. When several variables are highly correlated, it’s sometimes more informative to measure their importance together as a group. Or, a measure of variable importance such as Strobl et al.’s could be used instead.

#### Question

What happens when you add another predictor that is also highly correlated with V1?

#### Code

library(randomForest)  
  
simulated3 <- mutate(simulated2, "duplicate2" = simulated$V1 + rnorm(200) \* 0.1)  
model3 <- randomForest(y ~ ., data = simulated3,  
 importance = TRUE,  
 ntree = 1000)  
rfImp3 <- varImp(model3, scale = FALSE)  
rfImp3

## Overall  
## V1 4.91687329  
## V2 6.52816504  
## V3 0.58711552  
## V4 7.04870917  
## V5 2.03115561  
## V6 0.14213148  
## V7 0.10991985  
## V8 -0.08405687  
## V9 -0.01075028  
## V10 0.09230576  
## duplicate1 3.80068234  
## duplicate2 1.87721959

#### Response

Because duplicate2 is highly correlated with both V1 and duplicate1, adding the second duplicate predictor reduces the variable importance scores of both V1 and the first duplicate, duplicate1.

#### Question

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

#### Code

library(partykit)

## Loading required package: grid

## Loading required package: libcoin

## Loading required package: mvtnorm

#CI tree model for original data  
model4 <- cforest(y ~ ., data = simulated, ntree = 1000)  
#CI tree model with 1 additional uninformative variable  
model5 <- cforest(y ~ ., data = simulated2, ntree = 1000)  
#CI tree model with 2 additional uninformative variables  
model6 <- cforest(y ~ ., data = simulated3, ntree = 1000)  
#Variable importance scores with conditional = TRUE  
cfImp4.con <- varimp(model4, conditional = TRUE)  
cfImp5.con <- varimp(model5, conditional = TRUE)  
cfImp6.con <- varimp(model6, conditional = TRUE)  
#Variable importance scores with conditional = FALSE  
cfImp4.unc <- varimp(model4, conditional = FALSE)  
cfImp5.unc <- varimp(model5, conditional = FALSE)  
cfImp6.unc <- varimp(model6, conditional = FALSE)

## [1] "No add'l var's, conditional = TRUE"

## V1 V2 V3 V4 V5 V6   
## 6.43516035 5.09145324 0.13215164 5.96513447 1.34879237 -0.19143941   
## V7 V8 V9 V10   
## -0.07077046 -0.36577131 -0.32249962 -0.19569255

## [1] "1 add'l var's, conditional = TRUE"

## V1 V2 V3 V4 V5 V6   
## 3.14305172 4.96480823 0.07081723 5.72365409 1.31623662 -0.11512778   
## V7 V8 V9 V10 duplicate1   
## -0.14987141 -0.19642103 -0.21031583 -0.10791709 2.65165019

## [1] "2 add'l var's, conditional = TRUE"

## V1 V2 V3 V4 V5 V6   
## 2.327551509 4.844125813 -0.008135787 5.542279540 1.309393597 -0.146750782   
## V7 V8 V9 V10 duplicate1 duplicate2   
## -0.084207803 -0.258901852 -0.130949882 -0.154505095 2.182063275 0.826202282

## [1] "No add'l var's, conditional = FALSE"

## V1 V2 V3 V4 V5 V6   
## 8.366934415 5.900653256 0.204321211 7.225381451 2.119240114 -0.033185674   
## V7 V8 V9 V10   
## 0.139209377 -0.133312929 -0.002380075 -0.180685871

## [1] "1 add'l var's, conditional = FALSE"

## V1 V2 V3 V4 V5 V6   
## 6.12821975 5.83017936 0.18164133 6.56046590 1.98517223 -0.05859344   
## V7 V8 V9 V10 duplicate1   
## 0.06089181 -0.05375829 0.14633639 -0.10142300 5.76977589

## [1] "2 add'l var's, conditional = FALSE"

## V1 V2 V3 V4 V5 V6   
## 5.60814342 5.66703847 0.05579206 6.43689954 1.89592645 -0.08902450   
## V7 V8 V9 V10 duplicate1 duplicate2   
## 0.08690583 -0.06608416 0.12985785 -0.07452734 5.43495977 3.08408014

#### Response

As additional variables highly correlated to V1 were added to the model, the importance of V1 declined for both traditional and modified importance measures. For the traditional importance measure, the importance of V1 declined 51% and then 26% with one and two additional predictors, respectively. For the modified importance measure, the importance of V1 declined less: 27% with one additional variable, and an additional 8% when a second highly correlated variable was added to the model. Total decline in the importance of V1 under the traditional importance measure is 64%, and total decline in the importance of V1 under the modified importance measure is 33%. The modified importance measure mitigated the decline in variable importance associated with the addition of highly correlated predictors.

#### Question

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

#### Code

#Boosted tree model for original data  
library(gbm)

## Loaded gbm 2.1.8

model7 <- gbm(y ~ ., data = simulated, distribution = "gaussian")  
#Boosted tree model with 1 additional uninformative variable  
model8 <- gbm(y ~ ., data = simulated2, distribution = "gaussian")  
#Boosted tree model with 2 additional uninformative variables  
model9 <- gbm(y ~ ., data = simulated3, distribution = "gaussian")  
#Variable importance scores with method = relative.influence  
btImp7.ri <- summary.gbm(object = model7, method = relative.influence)

btImp8.ri <- summary.gbm(object = model8, method = relative.influence)

btImp9.ri <- summary.gbm(object = model9, method = relative.influence)

#Variable importance scores with method = permutation.test.gbm  
btImp7.pt <- summary.gbm(object = model7, method = permutation.test.gbm)

btImp8.pt <- summary.gbm(object = model8, method = permutation.test.gbm)

btImp9.pt <- summary.gbm(object = model9, method = permutation.test.gbm)

## [1] "No add'l var's, method = relative.importance"

## var rel.inf  
## V4 V4 29.2480954  
## V1 V1 27.8308275  
## V2 V2 23.5365573  
## V5 V5 10.8579622  
## V3 V3 7.9915978  
## V7 V7 0.2147969  
## V9 V9 0.1726843  
## V6 V6 0.1474784  
## V8 V8 0.0000000  
## V10 V10 0.0000000

## [1] "1 add'l var's, method = relative.importance"

## var rel.inf  
## V4 V4 28.3422124  
## V2 V2 24.0528986  
## duplicate1 duplicate1 14.6551646  
## V1 V1 13.7272625  
## V5 V5 10.8736058  
## V3 V3 8.1951176  
## V6 V6 0.1537385  
## V7 V7 0.0000000  
## V8 V8 0.0000000  
## V9 V9 0.0000000  
## V10 V10 0.0000000

## [1] "2 add'l var's, method = relative.importance"

## var rel.inf  
## V4 V4 31.5714557  
## V2 V2 22.0166944  
## V1 V1 13.6734623  
## duplicate1 duplicate1 13.4204025  
## V5 V5 10.2834297  
## V3 V3 8.3565726  
## V6 V6 0.3935196  
## V7 V7 0.1815789  
## duplicate2 duplicate2 0.1028842  
## V8 V8 0.0000000  
## V9 V9 0.0000000  
## V10 V10 0.0000000

## [1] "No add'l var's, method = relative.permutation.test.gbm"

## var rel.inf  
## 1 V4 36.57125254  
## 2 V1 26.17956859  
## 3 V2 21.65925303  
## 4 V5 9.12945991  
## 5 V3 6.30170169  
## 6 V7 0.06471367  
## 7 V6 0.05471134  
## 8 V9 0.03933923  
## 9 V8 0.00000000  
## 10 V10 0.00000000

## [1] "1 add'l var's, method = relative.permutation.test.gbm"

## var rel.inf  
## 1 V4 37.08826085  
## 2 V2 24.74377153  
## 3 V5 11.53116460  
## 4 V1 9.92921593  
## 5 duplicate1 9.44644702  
## 6 V3 7.19150181  
## 7 V6 0.06963826  
## 8 V7 0.00000000  
## 9 V8 0.00000000  
## 10 V9 0.00000000  
## 11 V10 0.00000000

## [1] "2 add'l var's, method = relative.permutation.test.gbm"

## var rel.inf  
## 1 V4 37.67891247  
## 2 V2 26.31089783  
## 3 V5 11.11713185  
## 4 V1 10.44633418  
## 5 V3 7.22686977  
## 6 duplicate1 6.85659656  
## 7 V6 0.26612477  
## 8 duplicate2 0.07438784  
## 9 V7 0.02274474  
## 10 V8 0.00000000  
## 11 V9 0.00000000  
## 12 V10 0.00000000

#### Response

For boosted trees, there are two methods for assessing variable importance: relative.influence, which is a traditional procedure, and an experimental procedure called permutation.test.gbm. As additional variables highly correlated with V1 are added to the model, the relative importance of V1 declines more steeply under the permutation.test.gbm method than under the relative.influence method. The importance of V1 declines a total of 51% under the relative importance measure when two additional variables are added. Under the permutation test, the total decline in importance for V1 is 60%.

#### Code

#Cubist Model  
library(Cubist)  
  
model10 <- cubist(x = simulated[-11], y = simulated$y, committees = 100)  
model11 <- cubist(x = simulated2[-11], y = simulated2$y, committees = 100)  
model12 <- cubist(x = simulated3[-11], y = simulated3$y, committees = 100)

#Note: output is truncated to emphasize measures of variable importance.  
summary(model10)  
##   
## Evaluation on training data (200 cases):  
##   
## Average |error| 1.480315  
## Relative |error| 0.37  
## Correlation coefficient 0.93  
##   
##   
## Attribute usage:  
## Conds Model  
##   
## 47% 96% V1  
## 40% 54% V3  
## 17% 100% V2  
## 96% V4  
## 66% V5  
## 26% V6  
##   
##   
## Time: 0.4 secs

summary(model11)

## Evaluation on training data (200 cases):  
##   
## Average |error| 1.462939  
## Relative |error| 0.36  
## Correlation coefficient 0.93  
##   
##   
## Attribute usage:  
## Conds Model  
##   
## 39% 48% V3  
## 38% 67% V1  
## 27% 92% V2  
## 2% 53% duplicate1  
## 92% V4  
## 8% V8  
## 54% V5  
## 20% V6  
## 2% V10  
##   
##   
## Time: 0.5 secs

summary(model12)

## Evaluation on training data (200 cases):  
##   
## Average |error| 1.354942  
## Relative |error| 0.34  
## Correlation coefficient 0.94  
##   
##   
## Attribute usage:  
## Conds Model  
##   
## 43% 58% V1  
## 38% 63% V3  
## 31% 95% V2  
## 41% duplicate2  
## 92% V4  
## 57% duplicate1  
## 7% V8  
## 57% V5  
## 17% V6  
##   
##   
## Time: 0.5 secs

#### Response

While “There is no established technique for measuring predictor importance for Cubist models” (KJ 212), we can observe the frequency with which a variable appeared in a rule criterion (“conds”). This gives some indication of the importance of a variable during the Cubist modeling process.

In the first model, which does not include additional variables correlated with V1, V1 appeared in rule criteria 47% of the time. As duplicate1 and then duplicate2 were added to the model, V1 appeared in rule criteria 38% and 43% of the time, respectively. This indicates some decline in variable importance for V1 as highly correlated variables are added to the model. In terms of its appearance in final models, V1 underwent a more significant decline: from 96% with no correlated duplicate variables, to 67%, and then to 58%.

# KJ 8.2

#### Question

Use a simulation to show tree bias with different granularities.

#### Code

#Create dataframe with some informative (x11, x12, x13) and some uninformative (x14, x15, x16) predictors.  
y <- rnorm(1000, 0, 1)  
x11 <- y + rnorm(1000, 1, 0.5)  
x12 <- y / 3 \* rnorm(1000, 0, 0.5)  
x13 <- (y + rnorm(1000, 0, 0.5)) ^ (1/3)   
x14 <- rnorm(1000, 0, 1)  
x15 <- runif(1000, 4, 6)  
x16 <- rexp(1000, 0.5)  
df\_1 <- data.frame(x11 = x11, x12 = x12, x13 = x13, x14 = x14, x15 = x15, x16 = x16, y = y)  
#Generate cutpoints for decreasing granularity of uninformative predictors (50, 20, and 10 distinct values).  
x14\_50 <- seq(from = min(df\_1$x14), to = max(df\_1$x14), length.out = 50)  
x14\_20 <- seq(from = min(df\_1$x14), to = max(df\_1$x14), length.out = 20)  
x14\_10 <- seq(from = min(df\_1$x14), to = max(df\_1$x14), length.out = 10)  
x15\_50 <- seq(from = min(df\_1$x15), to = max(df\_1$x15), length.out = 50)  
x15\_20 <- seq(from = min(df\_1$x15), to = max(df\_1$x15), length.out = 20)  
x15\_10 <- seq(from = min(df\_1$x15), to = max(df\_1$x15), length.out = 10)  
x16\_50 <- seq(from = min(df\_1$x16), to = max(df\_1$x16), length.out = 50)  
x16\_20 <- seq(from = min(df\_1$x16), to = max(df\_1$x16), length.out = 20)  
x16\_10 <- seq(from = min(df\_1$x16), to = max(df\_1$x16), length.out = 10)  
#Create low-granularity variables  
x24 <- as.numeric(as.vector(cut(df\_1$x14, breaks = 50, labels = x14\_50)))  
x34 <- as.numeric(as.vector(cut(df\_1$x14, breaks = 20, labels = x14\_20)))  
x44 <- as.numeric(as.vector(cut(df\_1$x14, breaks = 10, labels = x14\_10)))  
x25 <- as.numeric(as.vector(cut(df\_1$x15, breaks = 50, labels = x15\_50)))  
x35 <- as.numeric(as.vector(cut(df\_1$x15, breaks = 20, labels = x15\_20)))  
x45 <- as.numeric(as.vector(cut(df\_1$x15, breaks = 10, labels = x15\_10)))  
x26 <- as.numeric(as.vector(cut(df\_1$x16, breaks = 50, labels = x16\_50)))  
x36 <- as.numeric(as.vector(cut(df\_1$x16, breaks = 20, labels = x16\_20)))  
x46 <- as.numeric(as.vector(cut(df\_1$x16, breaks = 10, labels = x16\_10)))  
#Construct reduced granularity dataframes  
df\_2 <- data.frame(x11 = x11, x12 = x12, x13 = x13, x24 = x24, x25 = x25, x26 = x26, y = y)  
df\_3 <- data.frame(x11 = x11, x12 = x12, x13 = x13, x34 = x34, x35 = x35, x36 = x36, y = y)  
df\_4 <- data.frame(x11 = x11, x12 = x12, x13 = x13, x44 = x44, x45 = x45, x46 = x46, y = y)  
#Construct models  
library(rpart)  
  
df\_1.mod <- rpart(y ~ ., data = df\_1)  
df\_2.mod <- rpart(y ~ ., data = df\_2)  
df\_3.mod <- rpart(y ~ ., data = df\_3)  
df\_4.mod <- rpart(y ~ ., data = df\_4)

## [1] "Variable importance with high granularity uninformative vars"

## Overall  
## x11 2.55169623  
## x12 0.90264412  
## x13 1.24738927  
## x14 0.08509515  
## x15 0.13668230  
## x16 0.10924922

## [1] "Variable importance with uninformative vars (50 distinct vals)"

## Overall  
## x11 2.55169623  
## x12 0.90264412  
## x13 1.24738927  
## x24 0.06262627  
## x25 0.12793460  
## x26 0.07365423

## [1] "Variable importance with uninformative vars (20 distinct vals)"

## Overall  
## x11 2.55169623  
## x12 0.90264412  
## x13 1.24738927  
## x34 0.06487874  
## x35 0.10749465  
## x36 0.05263083

## [1] "Variable importance with uninformative vars (10 distinct vals"

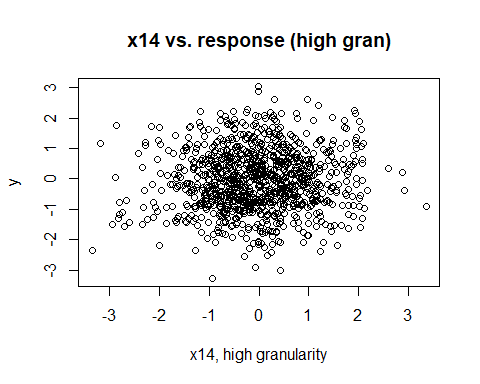
## Overall  
## x11 2.55169623  
## x12 0.90264412  
## x13 1.24738927  
## x44 0.03412582  
## x45 0.10152804  
## x46 0.04846781

#### Response

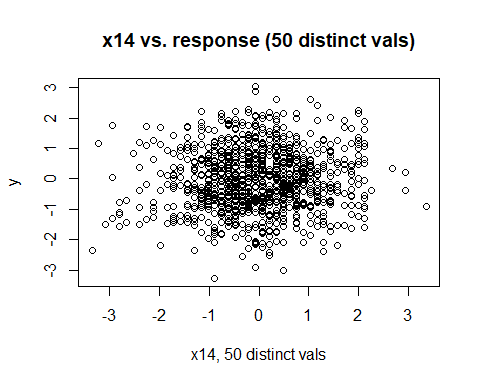
Tree bias due to granularity occurs when a variable’s importance in a tree model is inflated not due to its strong relationship to the response, but merely because of the large number of distinct values that variable takes on. Variables that take on large numbers of distinct values are said to be granular. To demonstrate tree bias due to granularity, we examined a single dataset comprised of some informative and some uninformative variables. By binning the uninformative variables (first to 50 distinct values, then 20, then 10) we were able to reduce their granularity. If bias due to granularity exists in the original high-granularity data, then the variable importance of the uninformative variables should decrease as they are binned to fewer and fewer distinct values.

The plots below show the relationship between X14, an uninformative variable, and the response variable, as the granularity of x14 is decreased.

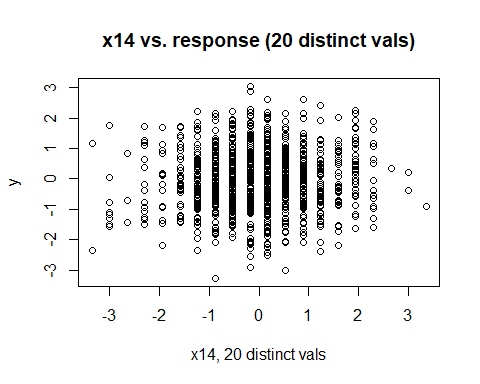
plot(df\_1$x14, df\_1$y, xlab = "x14, high granularity", ylab = "y", main = "x14 vs. response (high gran)")



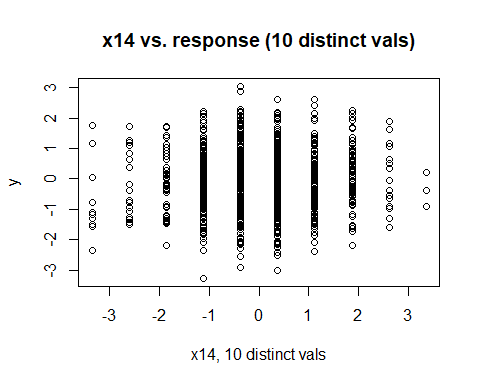
plot(df\_2$x24, df\_2$y, xlab = "x14, 50 distinct vals", ylab = "y", main = "x14 vs. response (50 distinct vals)")



plot(df\_3$x34, df\_3$y, xlab = "x14, 20 distinct vals", ylab = "y", main = "x14 vs. response (20 distinct vals)")



plot(df\_4$x44, df\_4$y, xlab = "x14, 10 distinct vals", ylab = "y", main = "x14 vs. response (10 distinct vals)")



The code in the **Code** section above shows the construction of the original dataset, along with the process for reducing the granularity of each uninformative variable through binning. This code is followed by variable importance tables for each model as granularity is varied.

As the granularity of the non-explanatory variables decreased, the overall importance of those variables decreased as well. This confirms that increased granularity can bias a tree model toward predictors with greater granularity, even when those predictors contain no information about the response variable.

# KJ 8.3

#### Question

In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by the gradi- ent. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 8.24 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data. The left-hand plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9:

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

#### Response

Bagging, short for bootstrap aggregation, is an ensemble technique that combines the predictions from multiple algorithms to make more accurate predictions than any individual model. Bagging is used to reduce variance of a decision tree.

Model on the right has both the bagging fraction and learning rate set at a higher value (0.9) compared to the model on the left (0.1). High learning rate typically results in overfit models and poor performance. High bagging fraction means a larger subset of the predictor variables are selected and the few very important predictors will stand out.

#### Question

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

#### Response

The left model with slow learning rate should be able to better predict the new data. The model with high learning rate can cause the model to converge to a suboptimal solution too quickly and result in overfitting.

#### Question

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

#### Response

Interaction depth controls the number of splits and the number of terminal nodes. In maximum depth, each tree contributes equally to the final model with the highest level of variable interactions. Increasing the depth will increase the spread of important variables, thus decreasing the slope.

# KJ 8.7

#### Question

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

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

#### Code

First, I need to bring in the data. I renamed it cmp so I didn’t have to continuously write ChemicalManufacturingProcess.

library(AppliedPredictiveModeling)  
  
data(ChemicalManufacturingProcess)  
  
cmp <- ChemicalManufacturingProcess

Next, I split the target variable off from the features and checked for missing values.

target <- cmp[,1]  
features <- cmp[,-1]  
  
summary(target)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 35.25 38.75 39.97 40.18 41.48 46.34

cc <- complete.cases(features)  
(length(cc[cc == TRUE])/length(cc))\*100

## [1] 86.36364

missing\_data <- sapply(features, function(x) sum(is.na(x)))  
missing\_data <- data.frame(missing\_data)  
(sum(missing\_data)/(176\*58))\*100

## [1] 1.038401

There were no missing values in the target (thankfully), but about 1% of the data was missing from the features. I decided to simply fill in those missing values with the mean of each feature since over 85% of the data was complete.

na\_to\_mean <- function(x) replace(x, is.na(x), mean(x, na.rm = TRUE))  
features[] <- lapply(features, na\_to\_mean)

The next step is to split the data in train and test sets.

set.seed(12345)  
train\_ind <- sample(seq\_len(nrow(features)),  
 size = floor(0.75\*nrow(features)))  
  
train\_x <- features[train\_ind,]  
test\_x <- features[-train\_ind,]  
  
train\_y <- target[train\_ind]  
test\_y <- target[-train\_ind]

Next, I need to build a couple tree-based models! I’m going to do single trees, random forest, and boosted!

# Single Tree  
library(rpart)  
  
rpartTree <- rpart(train\_y ~ .,  
 data = train\_x,  
 method = "anova")  
  
rpartPredict <- predict(rpartTree, test\_x)  
rmse <- sqrt(mean((test\_y - rpartPredict)^2))  
rmse

## [1] 1.328552

# Random Forest  
library(randomForest)  
  
rfModel <- randomForest(train\_x,  
 train\_y,  
 importance = TRUE,  
 ntress = 1000)  
  
rfPredict <- predict(rfModel, test\_x)  
rmse <- sqrt(mean((test\_y - rfPredict)^2))  
rmse

## [1] 1.106501

# Boosted  
library(gbm)  
  
gbmModel <- gbm.fit(train\_x,  
 train\_y,  
 distribution = "gaussian")

## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.2827 nan 0.0010 0.0011  
## 2 3.2807 nan 0.0010 0.0014  
## 3 3.2790 nan 0.0010 0.0013  
## 4 3.2764 nan 0.0010 0.0021  
## 5 3.2739 nan 0.0010 0.0020  
## 6 3.2722 nan 0.0010 0.0012  
## 7 3.2694 nan 0.0010 0.0023  
## 8 3.2674 nan 0.0010 0.0012  
## 9 3.2649 nan 0.0010 0.0019  
## 10 3.2624 nan 0.0010 0.0024  
## 20 3.2382 nan 0.0010 0.0020  
## 40 3.1927 nan 0.0010 0.0018  
## 60 3.1471 nan 0.0010 0.0020  
## 80 3.1068 nan 0.0010 0.0008  
## 100 3.0689 nan 0.0010 0.0019

gbmPredict <- predict(gbmModel, test\_x)

## Using 100 trees...

rmse <- sqrt(mean((test\_y - gbmPredict)^2))  
rmse

## [1] 1.86659

#### Response

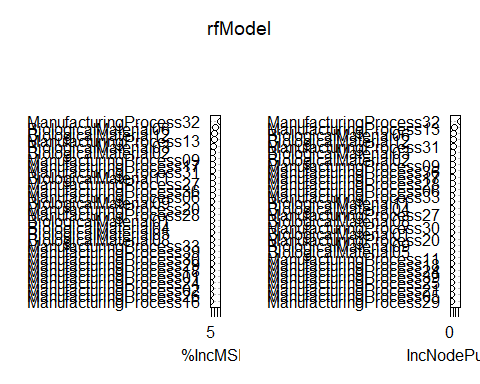
I created three different tree-based models: single tree, random forest, and boosted trees. Based on RMSE, the random forest model was the best one (lowestd RMSE).

#### Question

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?

#### Code

library(randomForest)  
  
# Look at best model  
varImpPlot(rfModel)



top\_predictors <- sort(rfModel$importance, decreasing = TRUE)[1:10]

#### Response

The most important predictors in this optimal model are MP32, MP13, BM06, and BM12. The top of the list seems to be pretty split between biological and process variables. Even if you look further down on the list, it stays pretty split the entire way down. This is similar to what was found by our other models in previous problems.

# Recommender System Problem

#### Question

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.

#### Code

library(arules)

## Loading required package: Matrix

##   
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack

##   
## Attaching package: 'arules'

## The following object is masked from 'package:kernlab':  
##   
## size

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

library(dplyr)

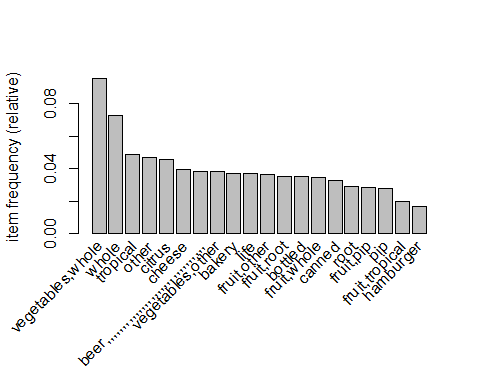
grocery <- read.transactions('https://raw.githubusercontent.com/dmoste/DATA624/main/HW2/GroceryDataSet.csv')

## Warning in asMethod(object): removing duplicated items in transactions

summary(grocery)

## transactions as itemMatrix in sparse format with  
## 9835 rows (elements/itemsets/transactions) and  
## 8219 columns (items) and a density of 0.0004422899   
##   
## most frequent items:  
## vegetables,whole whole tropical other   
## 940 717 482 460   
## citrus (Other)   
## 453 32700   
##   
## element (itemset/transaction) length distribution:  
## sizes  
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16   
## 1380 2733 1774 1257 910 601 415 293 166 95 75 44 39 19 11 9   
## 17 18 19 20 21 23   
## 2 3 3 3 1 2   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 2.000 3.000 3.635 5.000 23.000   
##   
## includes extended item information - examples:  
## labels  
## 1 (appetizer),,,,,,,,,,,,,,,,,,,,,  
## 2 (appetizer),,,,,,,,,,,,,,,,,,,,,,,  
## 3 (appetizer),,,,,,,,,,,,,,,,,,,,,,,,

itemFrequencyPlot(grocery, topN = 20)



crossTable(grocery, measure = "support", sort = TRUE)[1:5, 1:5]

## vegetables,whole whole tropical other citrus  
## vegetables,whole 0.095577021 0.0000000 0.006405694 0.01403152 0.006202339  
## whole 0.000000000 0.0729029 0.000000000 0.00000000 0.000000000  
## tropical 0.006405694 0.0000000 0.049008643 0.00000000 0.000000000  
## other 0.014031520 0.0000000 0.000000000 0.04677173 0.000000000  
## citrus 0.006202339 0.0000000 0.000000000 0.00000000 0.046059990

rules <- apriori(grocery, control = list(verbose = FALSE),  
 parameter = list(support = 0.001, confidence = 0.25,  
 minlen = 2))  
top\_lift <- sort(rules, by = "lift", decreasing = TRUE)[1:10]  
inspect(top\_lift)

## lhs rhs support confidence coverage lift count  
## [1] {plants,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} => {pot} 0.001016777 1.0000000 0.001016777 894.0909 10  
## [2] {pot} => {plants,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} 0.001016777 0.9090909 0.001118454 894.0909 10  
## [3] {articles,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} => {hygiene} 0.001220132 1.0000000 0.001220132 614.6875 12  
## [4] {hygiene} => {articles,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} 0.001220132 0.7500000 0.001626843 614.6875 12  
## [5] {product,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} => {long} 0.001118454 1.0000000 0.001118454 546.3889 11  
## [6] {long} => {product,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} 0.001118454 0.6111111 0.001830198 546.3889 11  
## [7] {bakery,   
## product,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} => {long} 0.001118454 1.0000000 0.001118454 546.3889 11  
## [8] {bakery,   
## long} => {product,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} 0.001118454 0.6111111 0.001830198 546.3889 11  
## [9] {life,   
## product,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} => {long} 0.001118454 1.0000000 0.001118454 546.3889 11  
## [10] {life,   
## long} => {product,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,} 0.001118454 0.6111111 0.001830198 546.3889 11

top\_support <- sort(rules, by = "support", decreasing = TRUE)[1:10]  
#inspect(top\_support)

#### Response

According to the algorithm, the top ten rules by lift are listed above. The top two rules involve people buying products from the bakery also buying life items. People also tend to buy fruits and vegetables together.