Homework Set 2 in R

DATA 624-01 Group 3

Z. Palmore, K. Popkin, K. Potter, C. Nan, J. Ramalingam

7/8/2021

Table of Contents

[KJ 6.3 2](#_Toc77316991)

[Part A 2](#_Toc77316992)

[Part B 5](#_Toc77316993)

[Part C 6](#_Toc77316994)

[Part D 10](#_Toc77316995)

[Part E 11](#_Toc77316996)

[Part F 12](#_Toc77316997)

[KJ 7.2 13](#_Toc77316998)

[Part A 14](#_Toc77316999)

[KJ 7.5 20](#_Toc77317000)

[Part A 20](#_Toc77317001)

[Part B 23](#_Toc77317002)

[Part C 24](#_Toc77317003)

[KJ 8.1 26](#_Toc77317004)

[Part A 26](#_Toc77317005)

[Part B 27](#_Toc77317006)

[Part C 28](#_Toc77317007)

[Part D 30](#_Toc77317008)

[KJ 8.2 32](#_Toc77317009)

[Part A 32](#_Toc77317010)

[KJ 8.3 33](#_Toc77317011)

[Part A 34](#_Toc77317012)

[Part B 34](#_Toc77317013)

[Part C 34](#_Toc77317014)

[KJ 8.7 36](#_Toc77317015)

[Part A 36](#_Toc77317016)

[Part B 40](#_Toc77317017)

[Part C 47](#_Toc77317018)

[Market Basket Analysis 48](#_Toc77317019)

## KJ 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:

### Part A

#### Question

Start R and use these commands to load the data:

> library(AppliedPredictiveModeling)  
> data(chemicalManufacturingProcess)

#### Code

library(AppliedPredictiveModeling)  
library(flextable)  
library(dplyr)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
# Show initial observations  
initial.obs <- as.data.frame(t(head(df)))  
initial.obs <- initial.obs %>%   
 mutate(Variable = colnames(df)) %>%   
 dplyr::select(Variable, 1, 2, 3, 4, 5)  
flextable(initial.obs) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Variable** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- |
| Yield | 38.000 | 42.440 | 42.030 | 41.420 | 42.490 |
| BiologicalMaterial01 | 6.250 | 8.010 | 8.010 | 8.010 | 7.470 |
| BiologicalMaterial02 | 49.580 | 60.970 | 60.970 | 60.970 | 63.330 |
| BiologicalMaterial03 | 56.970 | 67.480 | 67.480 | 67.480 | 72.250 |
| BiologicalMaterial04 | 12.740 | 14.650 | 14.650 | 14.650 | 14.020 |
| BiologicalMaterial05 | 19.510 | 19.360 | 19.360 | 19.360 | 17.910 |
| BiologicalMaterial06 | 43.730 | 53.140 | 53.140 | 53.140 | 54.660 |
| BiologicalMaterial07 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 |
| BiologicalMaterial08 | 16.660 | 19.040 | 19.040 | 19.040 | 18.220 |
| BiologicalMaterial09 | 11.440 | 12.550 | 12.550 | 12.550 | 12.800 |
| BiologicalMaterial10 | 3.460 | 3.460 | 3.460 | 3.460 | 3.050 |
| BiologicalMaterial11 | 138.090 | 153.670 | 153.670 | 153.670 | 147.610 |
| BiologicalMaterial12 | 18.830 | 21.050 | 21.050 | 21.050 | 21.050 |
| ManufacturingProcess01 |  | 0.000 | 0.000 | 0.000 | 10.700 |
| ManufacturingProcess02 |  | 0.000 | 0.000 | 0.000 | 0.000 |
| ManufacturingProcess03 |  |  |  |  |  |
| ManufacturingProcess04 |  | 917.000 | 912.000 | 911.000 | 918.000 |
| ManufacturingProcess05 |  | 1,032.200 | 1,003.600 | 1,014.600 | 1,027.500 |
| ManufacturingProcess06 |  | 210.000 | 207.100 | 213.300 | 205.700 |
| ManufacturingProcess07 |  | 177.000 | 178.000 | 177.000 | 178.000 |
| ManufacturingProcess08 |  | 178.000 | 178.000 | 177.000 | 178.000 |
| ManufacturingProcess09 | 43.000 | 46.570 | 45.070 | 44.920 | 44.960 |
| ManufacturingProcess10 |  |  |  |  |  |
| ManufacturingProcess11 |  |  |  |  |  |
| ManufacturingProcess12 |  | 0.000 | 0.000 | 0.000 | 0.000 |
| ManufacturingProcess13 | 35.500 | 34.000 | 34.800 | 34.800 | 34.600 |
| ManufacturingProcess14 | 4,898.000 | 4,869.000 | 4,878.000 | 4,897.000 | 4,992.000 |
| ManufacturingProcess15 | 6,108.000 | 6,095.000 | 6,087.000 | 6,102.000 | 6,233.000 |
| ManufacturingProcess16 | 4,682.000 | 4,617.000 | 4,617.000 | 4,635.000 | 4,733.000 |
| ManufacturingProcess17 | 35.500 | 34.000 | 34.800 | 34.800 | 33.900 |
| ManufacturingProcess18 | 4,865.000 | 4,867.000 | 4,877.000 | 4,872.000 | 4,886.000 |
| ManufacturingProcess19 | 6,049.000 | 6,097.000 | 6,078.000 | 6,073.000 | 6,102.000 |
| ManufacturingProcess20 | 4,665.000 | 4,621.000 | 4,621.000 | 4,611.000 | 4,659.000 |
| ManufacturingProcess21 | 0.000 | 0.000 | 0.000 | 0.000 | -0.700 |
| ManufacturingProcess22 |  | 3.000 | 4.000 | 5.000 | 8.000 |
| ManufacturingProcess23 |  | 0.000 | 1.000 | 2.000 | 4.000 |
| ManufacturingProcess24 |  | 3.000 | 4.000 | 5.000 | 18.000 |
| ManufacturingProcess25 | 4,873.000 | 4,869.000 | 4,897.000 | 4,892.000 | 4,930.000 |
| ManufacturingProcess26 | 6,074.000 | 6,107.000 | 6,116.000 | 6,111.000 | 6,151.000 |
| ManufacturingProcess27 | 4,685.000 | 4,630.000 | 4,637.000 | 4,630.000 | 4,684.000 |
| ManufacturingProcess28 | 10.700 | 11.200 | 11.100 | 11.100 | 11.300 |
| ManufacturingProcess29 | 21.000 | 21.400 | 21.300 | 21.300 | 21.600 |
| ManufacturingProcess30 | 9.900 | 9.900 | 9.400 | 9.400 | 9.000 |
| ManufacturingProcess31 | 69.100 | 68.700 | 69.300 | 69.300 | 69.400 |
| ManufacturingProcess32 | 156.000 | 169.000 | 173.000 | 171.000 | 171.000 |
| ManufacturingProcess33 | 66.000 | 66.000 | 66.000 | 68.000 | 70.000 |
| ManufacturingProcess34 | 2.400 | 2.600 | 2.600 | 2.500 | 2.500 |
| ManufacturingProcess35 | 486.000 | 508.000 | 509.000 | 496.000 | 468.000 |
| ManufacturingProcess36 | 0.019 | 0.019 | 0.018 | 0.018 | 0.017 |
| ManufacturingProcess37 | 0.500 | 2.000 | 0.700 | 1.200 | 0.200 |
| ManufacturingProcess38 | 3.000 | 2.000 | 2.000 | 2.000 | 2.000 |
| ManufacturingProcess39 | 7.200 | 7.200 | 7.200 | 7.200 | 7.300 |
| ManufacturingProcess40 |  | 0.100 | 0.000 | 0.000 | 0.000 |
| ManufacturingProcess41 |  | 0.150 | 0.000 | 0.000 | 0.000 |
| ManufacturingProcess42 | 11.600 | 11.100 | 12.000 | 10.600 | 11.000 |
| ManufacturingProcess43 | 3.000 | 0.900 | 1.000 | 1.100 | 1.100 |
| ManufacturingProcess44 | 1.800 | 1.900 | 1.800 | 1.800 | 1.700 |
| ManufacturingProcess45 | 2.400 | 2.200 | 2.300 | 2.100 | 2.100 |

#### Response

Our code displays a table with the first 5 observations of each predictor variable. The table contains 57 predictors with 12 of those describing the input biological material and 45 describing the process predictors from 176 manufacturing runs. The response variable ‘Yield’ contains the percent yield for each run. There are missing values for several variables that will need imputation.

### Part B

#### Question

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

library(mice)  
# Find what percentage of the data contain missing values  
length(df$Yield) # 176   
sum(is.na(df)) # 106  
sum(is.na(df))/(length(df$Yield) \* length(df)) \* 100 # ~1.04%   
# Check for patterns in missing data  
md.pattern(df) # They are random - data is 'observed'  
# Using mice function for predictive mean matching  
df.mice <- mice(df, method = "pmm", seed = 624)  
summary(df.mice)

# Method using median of each feature  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}

#### Response

Roughly 1.04% of the data was missing. It was first checked for patterns in the missing values. There were none. The missing data points were noted as random occurrences where no special treatment was required during imputation. We then ran two methods of imputation for referencing, one by predictive mean matching and the other by median. We decided to use the data set imputed by the median of each variable. No missing values remained.

### Part C

#### Question

Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter.

#### Code

library(AppliedPredictiveModeling)  
library(reshape2)  
library(dplyr)  
library(caret)  
library(Metrics)  
df.nz <- df %>%   
 select\_at(vars(-one\_of(nearZeroVar(.,names=T))))  
cormat <- round(cor(df.nz),2)  
melted\_cormat = melt(cormat)  
melted\_cormat\_df = data.frame(melted\_cormat)  
#Filter to only Yield and sort in descending order  
yield\_corr <- melted\_cormat\_df %>%   
 filter(Var2 == "Yield")  
yield\_corr$absvalue = abs(yield\_corr$value)  
yield\_corr2 = yield\_corr[order(-yield\_corr[,4]),]

set.seed(41)  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
  
  
#Create the regression model  
lmyield = lm(Yield~ManufacturingProcess32 +   
 ManufacturingProcess36 +   
 ManufacturingProcess09 +   
 ManufacturingProcess13 +   
 BiologicalMaterial02 +   
 BiologicalMaterial06 +   
 BiologicalMaterial03,   
 data=train)  
lmmod.summary <- summary(lmyield)  
lmmod.summary

##   
## Call:  
## lm(formula = Yield ~ ManufacturingProcess32 + ManufacturingProcess36 +   
## ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 +   
## BiologicalMaterial06 + BiologicalMaterial03, data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2.82742 -0.98950 0.03976 0.73478 2.97533   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 15.13509 14.27099 1.061 0.290956   
## ManufacturingProcess32 0.13746 0.03552 3.870 0.000175 \*\*\*  
## ManufacturingProcess36 -270.45050 197.99212 -1.366 0.174422   
## ManufacturingProcess09 0.40979 0.12412 3.302 0.001256 \*\*   
## ManufacturingProcess13 -0.32801 0.18695 -1.755 0.081806 .   
## BiologicalMaterial02 -0.06963 0.09139 -0.762 0.447579   
## BiologicalMaterial06 0.06658 0.09804 0.679 0.498356   
## BiologicalMaterial03 0.02689 0.05577 0.482 0.630577   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.254 on 124 degrees of freedom  
## Multiple R-squared: 0.5714, Adjusted R-squared: 0.5472   
## F-statistic: 23.62 on 7 and 124 DF, p-value: < 2.2e-16

# Check the RMSE and R2  
lmyield\_train = predict(lmyield, newdata = train, interval ='prediction')  
lmyield\_train\_df = data.frame(lmyield\_train)  
lmyield\_train\_df$actual = train$Yield  
lmyield\_train\_rmse = rmse(lmyield\_train\_df$actual, lmyield\_train\_df$fit)  
df.rtrain<- data.frame(RMSE = lmyield\_train\_rmse, R2 = caret::R2(lmyield\_train, train$Yield))  
df.rtrain

## RMSE R2.fit R2.lwr R2.upr  
## 1 1.215335 0.5714077 0.5658518 0.575571

#### Response

For pre-processing we identified and removed variables with a near zero variance then converted our existing data frame into a long format. This process removed one variable, “BiologicalMaterial07.” We then created a correlation matrix of the predictors with our response, ‘Yield,’ and down selected to reduce the data dimensions and complete our preprocessing steps.

After splitting the data 75% to training and 25% to our test set, we chose to build a linear regression model using the top 7 features that had the strongest correlations with ‘Yield.’ They were selected based on absolute correlation strength. A table of all variable correlations with Yield is shown for reference.

| **Var1** | **Var2** | **value** | **absvalue** |
| --- | --- | --- | --- |
| Yield | Yield | 1.00 | 1.00 |
| ManufacturingProcess32 | Yield | 0.61 | 0.61 |
| ManufacturingProcess36 | Yield | -0.53 | 0.53 |
| ManufacturingProcess09 | Yield | 0.50 | 0.50 |
| ManufacturingProcess13 | Yield | -0.50 | 0.50 |
| BiologicalMaterial02 | Yield | 0.48 | 0.48 |
| BiologicalMaterial06 | Yield | 0.48 | 0.48 |
| BiologicalMaterial03 | Yield | 0.45 | 0.45 |
| ManufacturingProcess17 | Yield | -0.43 | 0.43 |
| ManufacturingProcess33 | Yield | 0.42 | 0.42 |
| ManufacturingProcess06 | Yield | 0.39 | 0.39 |
| BiologicalMaterial04 | Yield | 0.38 | 0.38 |
| BiologicalMaterial08 | Yield | 0.38 | 0.38 |
| BiologicalMaterial12 | Yield | 0.37 | 0.37 |
| BiologicalMaterial01 | Yield | 0.36 | 0.36 |
| BiologicalMaterial11 | Yield | 0.35 | 0.35 |
| ManufacturingProcess12 | Yield | 0.35 | 0.35 |
| ManufacturingProcess11 | Yield | 0.32 | 0.32 |
| ManufacturingProcess04 | Yield | -0.26 | 0.26 |
| ManufacturingProcess28 | Yield | 0.26 | 0.26 |
| ManufacturingProcess30 | Yield | 0.23 | 0.23 |
| ManufacturingProcess15 | Yield | 0.22 | 0.22 |
| ManufacturingProcess02 | Yield | -0.21 | 0.21 |
| ManufacturingProcess24 | Yield | -0.21 | 0.21 |
| BiologicalMaterial10 | Yield | 0.20 | 0.20 |
| ManufacturingProcess10 | Yield | 0.20 | 0.20 |
| ManufacturingProcess34 | Yield | 0.17 | 0.17 |
| ManufacturingProcess35 | Yield | -0.17 | 0.17 |
| ManufacturingProcess37 | Yield | -0.16 | 0.16 |
| ManufacturingProcess43 | Yield | 0.16 | 0.16 |
| BiologicalMaterial05 | Yield | 0.15 | 0.15 |
| ManufacturingProcess29 | Yield | 0.15 | 0.15 |
| ManufacturingProcess19 | Yield | 0.13 | 0.13 |
| ManufacturingProcess05 | Yield | 0.11 | 0.11 |
| ManufacturingProcess01 | Yield | -0.10 | 0.10 |
| ManufacturingProcess23 | Yield | -0.10 | 0.10 |
| BiologicalMaterial09 | Yield | 0.09 | 0.09 |
| ManufacturingProcess03 | Yield | -0.09 | 0.09 |
| ManufacturingProcess38 | Yield | -0.09 | 0.09 |
| ManufacturingProcess20 | Yield | -0.07 | 0.07 |
| ManufacturingProcess31 | Yield | -0.07 | 0.07 |
| ManufacturingProcess44 | Yield | 0.07 | 0.07 |
| ManufacturingProcess18 | Yield | -0.06 | 0.06 |
| ManufacturingProcess07 | Yield | -0.04 | 0.04 |
| ManufacturingProcess16 | Yield | -0.04 | 0.04 |
| ManufacturingProcess26 | Yield | 0.04 | 0.04 |
| ManufacturingProcess39 | Yield | 0.04 | 0.04 |
| ManufacturingProcess40 | Yield | -0.04 | 0.04 |
| ManufacturingProcess21 | Yield | -0.03 | 0.03 |
| ManufacturingProcess41 | Yield | -0.03 | 0.03 |
| ManufacturingProcess45 | Yield | 0.03 | 0.03 |
| ManufacturingProcess22 | Yield | 0.02 | 0.02 |
| ManufacturingProcess08 | Yield | 0.01 | 0.01 |
| ManufacturingProcess14 | Yield | -0.01 | 0.01 |
| ManufacturingProcess25 | Yield | 0.01 | 0.01 |
| ManufacturingProcess42 | Yield | -0.01 | 0.01 |
| ManufacturingProcess27 | Yield | 0.00 | 0.00 |

Those selected are shown in the model summary which had a multiple R2 value of 0.5714077 indicating the model captures roughly 57% of the variation in response. We note that, this is likely not the best model for prediction with this data and the only significant predictors (based on an alpha level of 0.05) were ManufacturingProcess32 and ManufacturingProcess09. However, this was a simple model choice we could gather information from and improve on as necessary since we were given the option to tune the model of our choice.

### Part D

#### Question

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?

#### Code

library(Metrics)  
library(caret)  
#Predict with the test data  
lmyield\_test = predict(lmyield, newdata = test, interval ='prediction')  
lmyield\_test\_df = data.frame(lmyield\_test)  
lmyield\_test\_df$actual = test$Yield  
lmyield\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
cat('RMSE of the test data for this model is', lmyield\_test\_rmse)

## RMSE of the test data for this model is 0.8794943

# Show calculations as data frame with RMSE and R squared  
df.rtest <- data.frame(RMSE = lmyield\_test\_rmse, R2 = caret::R2(lmyield\_test, test$Yield))  
df.res <- rbind(df.rtrain, df.rtest)  
df.res <- df.res %>%   
 mutate(Dataset = c("Train", "Test")) %>%   
 dplyr::select(Dataset, RMSE, R2.fit, R2.lwr, R2.upr)  
flextable(df.res) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Dataset** | **RMSE** | **R2.fit** | **R2.lwr** | **R2.upr** |
| --- | --- | --- | --- | --- |
| Train | 1.2153355 | 0.5714077 | 0.5658518 | 0.5755710 |
| Test | 0.8794943 | 0.7868231 | 0.7840575 | 0.7888425 |

df.res$R2.fit[2]

## [1] 0.7868231

round(df.res$R2.fit[2],2)\*100

## [1] 79

#### Response

Our fit value is 0.7868231 which indicates our model explains about 79% on the test set. This is lower than optimal if we were making predictions based on it and our RMSE (root mean squared error) does not perform as well as it could either. Compared to the same metrics on the training dataset, the test data performed slightly poorer at prediction. We can trace these results back to the model selection, which, if reselected, would likely improve given the information we have gained from this model.

### Part E

#### Question

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

#### Code

library(caret)  
library(dplyr)  
var.importance <- varImp(lmyield)  
sort(var.importance$Overall, decreasing = TRUE)

## [1] 3.8700098 3.3016486 1.7545514 1.3659660 0.7618781 0.6790745 0.4821116

var.importance <- varImp(lmyield) %>%   
 tibble::rownames\_to\_column("Variable") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,3)) %>%   
 dplyr::select(Rank, Variable, Importance)  
flextable(var.importance) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Rank** | **Variable** | **Importance** |
| --- | --- | --- |
| 1 | ManufacturingProcess32 | 3.870 |
| 2 | ManufacturingProcess09 | 3.302 |
| 3 | ManufacturingProcess13 | 1.755 |
| 4 | ManufacturingProcess36 | 1.366 |
| 5 | BiologicalMaterial02 | 0.762 |
| 6 | BiologicalMaterial06 | 0.679 |
| 7 | BiologicalMaterial03 | 0.482 |

#### Response

As shown in the above table with rank, variable, and importance values, the manufacturing process takes the most importance. Process predictors are seemingly more important with our model. However, looking back at the correlations when down selecting for our model, manufacturing processes still dominated the list when considering all available variables.

Recall that since our model relied on the 7 strongest correlations with Yield, the number of variables used in the model were already reduced. We also note that only two of our variables had significance below an alpha level of 0.05. Both of those variables were also process predictors (ManufacturingProcess32 and ManufacturingProcess09).

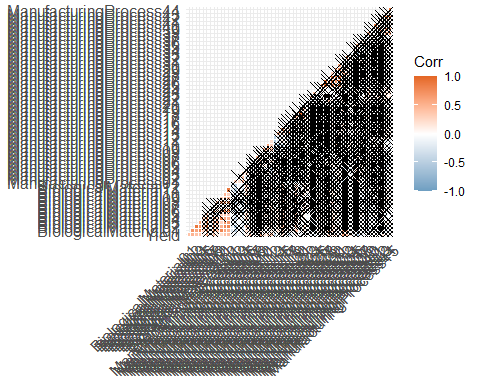
### Part F

#### Question

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

library(AppliedPredictiveModeling)  
library(ggcorrplot)  
library(corrplot)  
corr <- round(cor(df), 1)  
p.mat <- ggcorrplot::cor\_pmat(corr)  
ggcorrplot(corr = corr,   
 "square",   
 "upper",  
 p.mat = p.mat,   
 sig.level = 0.05,  
 outline.color = "white",   
 color = c("#6D9EC1", "white",  
 "#E46726"),  
 ggtheme = ggplot2::theme\_minimal()) + coord\_flip()



#### Response

We outline the relationships between each variable from the entire data set in the correlation plot where orange indicates a stronger positive relationship and blue indicates a stronger negative relationship.

Given our previous exploration of correlations, p-value significance levels, and feature fit with linear modeling, we can easily confirm that process predictors have more relevance in predicting yield. This bodes well as these features can be changes while biological ones cannot change. Simulations could be done using linear and other model types with modifications to focus on those predictors.

A good next step might be to build the model via forward stepwise or backward stepwise regression (or both). Training on some other models such as Random Forest, could also be used to identify which features are most relevant and expose their true nature when influencing yield. We note that prior to experimenting with process features value changes, it is difficult to know what will perform well.

## KJ 7.2

Friedman (1991) introduced several benchmark data sets created 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).The package mlbench contains a function called mlbench.friedman1 that simulates these data:

> library(mlbench)  
> 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)

### Part A

#### Question

Tune several models on these data. An example is shown in the code. Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)?

#### Code

library(mlbench)  
library(caret)  
library(AppliedPredictiveModeling)  
library(kernlab)  
library(doParallel) # Used for computation  
library(earth) # Package necessary for marsModel  
registerDoParallel(cores=2)  
getDoParWorkers()

## [1] 2

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)  
knnModel <- train(x = trainingData$x, y = trainingData$y, method = "knn", preProc = c("center", "scale"), tuneLength = 10)  
knnPred <- predict(knnModel, newdata = testData$x)  
## The function 'postResample' can be used to get the test set   
## performance values  
postResample(pred = knnPred, obs = testData$y)

## RMSE Rsquared MAE   
## 3.2040595 0.6819919 2.5683461

svmModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "svmRadial",  
 tuneLength=10,  
 preProc = c("center", "scale"))  
svmPred <- predict(svmModel, newdata = testData$x)  
postResample(pred = svmPred, obs = testData$y)

## RMSE Rsquared MAE   
## 2.0864652 0.8236735 1.5854649

nnetGrid <- expand.grid(.decay=c(0, 0.01, 0.1, 0.5, 0.9),  
 .size=c(1, 10, 15, 20),  
 .bag=FALSE)  
nnetModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "avNNet",  
 tuneGrid = nnetGrid,  
 preProc = c("center", "scale"),  
 trace=FALSE,  
 linout=TRUE,  
 maxit=500)  
# Neural net may take several minutes  
nnetPred <- predict(nnetModel, newdata = testData$x)  
postResample(pred = nnetPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.894755 0.856176 1.441820

marsGrid <- expand.grid(.degree=1:2,  
 .nprune=2:20)  
marsModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "earth",  
 tuneGrid = marsGrid,  
 preProc = c("center", "scale"))  
marsPred <- predict(marsModel, newdata = testData$x)  
postResample(pred = marsPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.3227340 0.9291489 1.0524686

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.

svmModel

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 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:  
##   
## C RMSE Rsquared MAE   
## 0.25 2.545335 0.7804647 2.015121  
## 0.50 2.319786 0.7965148 1.830009  
## 1.00 2.188349 0.8119636 1.726027  
## 2.00 2.103655 0.8241314 1.655842  
## 4.00 2.066879 0.8294322 1.631051  
## 8.00 2.052681 0.8313929 1.623550  
## 16.00 2.049867 0.8318312 1.621820  
## 32.00 2.049867 0.8318312 1.621820  
## 64.00 2.049867 0.8318312 1.621820  
## 128.00 2.049867 0.8318312 1.621820  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.06802164  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were sigma = 0.06802164 and C = 16.

nnetModel

## Model Averaged Neural Network   
##   
## 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:  
##   
## decay size RMSE Rsquared MAE   
## 0.00 1 2.589902 0.7324403 2.014450  
## 0.00 10 3.082909 0.6581005 2.345346  
## 0.00 15 2.666684 0.7202739 2.135432  
## 0.00 20 2.634394 0.7273102 2.116201  
## 0.01 1 2.567190 0.7363972 1.993021  
## 0.01 10 2.714509 0.7118994 2.173845  
## 0.01 15 2.435366 0.7634558 1.935705  
## 0.01 20 2.346376 0.7801847 1.850974  
## 0.10 1 2.580129 0.7336990 2.000459  
## 0.10 10 2.528971 0.7492960 2.003431  
## 0.10 15 2.309856 0.7879857 1.823430  
## 0.10 20 2.289300 0.7922572 1.799799  
## 0.50 1 2.620985 0.7251648 2.034073  
## 0.50 10 2.389468 0.7734132 1.893293  
## 0.50 15 2.248817 0.7979988 1.778851  
## 0.50 20 2.257951 0.7973906 1.768133  
## 0.90 1 2.649162 0.7195330 2.057453  
## 0.90 10 2.339031 0.7803865 1.849270  
## 0.90 15 2.247236 0.7980673 1.774157  
## 0.90 20 2.248629 0.7989807 1.770371  
##   
## Tuning parameter 'bag' was held constant at a value of FALSE  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were size = 15, decay = 0.9 and bag = FALSE.

marsModel

## Multivariate Adaptive Regression Spline   
##   
## 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:  
##   
## degree nprune RMSE Rsquared MAE   
## 1 2 4.420600 0.2049915 3.619816  
## 1 3 3.701246 0.4408654 2.951090  
## 1 4 2.799544 0.6800991 2.225051  
## 1 5 2.538474 0.7359840 2.028529  
## 1 6 2.392972 0.7670752 1.895457  
## 1 7 1.997878 0.8371559 1.571951  
## 1 8 1.876575 0.8570992 1.467108  
## 1 9 1.812964 0.8663208 1.420100  
## 1 10 1.782912 0.8704084 1.399319  
## 1 11 1.771684 0.8718386 1.386223  
## 1 12 1.787483 0.8701198 1.395825  
## 1 13 1.805686 0.8678252 1.407169  
## 1 14 1.804984 0.8683606 1.410113  
## 1 15 1.831547 0.8648770 1.433968  
## 1 16 1.839945 0.8637020 1.438650  
## 1 17 1.834362 0.8644089 1.434148  
## 1 18 1.834362 0.8644089 1.434148  
## 1 19 1.834362 0.8644089 1.434148  
## 1 20 1.834362 0.8644089 1.434148  
## 2 2 4.424916 0.2035789 3.628878  
## 2 3 3.680817 0.4479081 2.941723  
## 2 4 2.858745 0.6646861 2.265031  
## 2 5 2.589553 0.7239981 2.070119  
## 2 6 2.464649 0.7530842 1.944510  
## 2 7 2.090644 0.8201211 1.642564  
## 2 8 1.917597 0.8491333 1.508119  
## 2 9 1.808609 0.8641337 1.419806  
## 2 10 1.697999 0.8803917 1.328664  
## 2 11 1.600188 0.8932347 1.251957  
## 2 12 1.554919 0.8988995 1.222037  
## 2 13 1.482903 0.9082352 1.153521  
## 2 14 1.529860 0.9025187 1.186956  
## 2 15 1.551420 0.9004952 1.202656  
## 2 16 1.564789 0.8995960 1.211800  
## 2 17 1.555896 0.9007856 1.208746  
## 2 18 1.553024 0.9011837 1.207791  
## 2 19 1.566099 0.8996463 1.214404  
## 2 20 1.566099 0.8996463 1.214404  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were nprune = 13 and degree = 2.

pr.nnet <- postResample(pred = nnetPred, obs = testData$y)  
postResample(pred = nnetPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.894755 0.856176 1.441820

pr.svm <- postResample(pred = svmPred, obs = testData$y)  
postResample(pred = svmPred, obs = testData$y)

## RMSE Rsquared MAE   
## 2.0864652 0.8236735 1.5854649

pr.mars <- postResample(pred = marsPred, obs = testData$y)  
postResample(pred = marsPred, obs = testData$y)

## RMSE Rsquared MAE   
## 1.3227340 0.9291489 1.0524686

modstat7.2 <- as.data.frame(rbind(pr.nnet, pr.svm, pr.mars))  
modstat7.2 <- modstat7.2 %>%  
 mutate(Model = c("NNET", "SVM", "MARS")) %>%   
 dplyr::select(Model,RMSE,Rsquared,MAE)  
varImp(marsModel)

## earth variable importance  
##   
## Overall  
## X1 100.00  
## X4 75.40  
## X2 49.00  
## X5 15.72  
## X3 0.00

#### Response

KNN, SVM, Neural Network, and MARS models were used for testing performances. Since the KNN model code was given as an example and did not perform as well as the others, we leave its performance statistics out.

Based on RMSE, the MARS model performs best. It also appears to select the informative predictors (X1-X5). A table of these statistics is provided to compare each model we created.

| **Model** | **RMSE** | **Rsquared** | **MAE** |
| --- | --- | --- | --- |
| NNET | 1.894755 | 0.8561760 | 1.441820 |
| SVM | 2.086465 | 0.8236735 | 1.585465 |
| MARS | 1.322734 | 0.9291489 | 1.052469 |

## KJ 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.

### Part A

#### Question

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

#### Code

# Recreate RMSE from 6.3  
########################  
# Part A load the data  
library(AppliedPredictiveModeling)  
library(reshape2)  
library(dplyr)  
library(Metrics)  
library(caret)  
set.seed(0701)  
# Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
# Part B Fill in missing values with the median of each feature  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}  
# Part C Correlations and Modeling  
cormat <- round(cor(df),2)  
melted\_cormat = melt(cormat)  
melted\_cormat\_df = data.frame(melted\_cormat)  
# Filter to only Yield and sort in descending order  
yield\_corr <- melted\_cormat\_df %>%   
 filter(Var2 == "Yield")  
yield\_corr$absvalue = abs(yield\_corr$value)  
yield\_corr2 = yield\_corr[order(-yield\_corr[,4]),]  
# Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
#Create the regression model  
lmyield = lm(Yield~ManufacturingProcess32 +   
 ManufacturingProcess36 +   
 ManufacturingProcess09 +   
 ManufacturingProcess13 +   
 BiologicalMaterial02 +   
 BiologicalMaterial06 +   
 BiologicalMaterial03,   
 data=train)  
# Part D Predictions  
# Predict with the test data  
lmyield\_test = predict(lmyield, newdata = test, interval ='prediction')  
lmyield\_test\_df = data.frame(lmyield\_test)  
lmyield\_test\_df$actual = test$Yield  
lmyield\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)

# Apply 6.3 to 7.5   
########################  
# Create Train and Test data sets  
set.seed(0701)  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
# Create the train x and train y datasets for HW 7.5  
trainx = train[,2:ncol(train)]  
trainy = train$Yield  
testx = test[,2:ncol(test)]  
testy = test$Yield  
# K-Nearest Neighbors (KNN) Model  
mod.knn <- train(x = trainx,   
 y = trainy,   
 method = "knn",  
 tuneLength=10)  
# Radial Supper Vector Machine (SVM) Model  
mod.svm <- train(x = trainx,  
 y = trainy,  
 method = "svmRadial",  
 tuneLength=10)  
# Multivariate Adaptive Regression Spline (MARS) Model  
marsGrid <- expand.grid(.degree=1:2,   
 .nprune=2:10)  
mod.mar <- train(x = trainx,   
 y = trainy,   
 method = "earth",  
 tuneGrid = marsGrid)  
# Tabulate results  
mod.compare <- resamples(list(  
 KNN = mod.knn, SVM = mod.svm, MARS = mod.mar))  
summary(mod.compare)

##   
## Call:  
## summary.resamples(object = mod.compare)  
##   
## Models: KNN, SVM, MARS   
## Number of resamples: 25   
##   
## MAE   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## KNN 1.077845 1.2670185 1.3475627 1.3448717 1.385247 1.542073 0  
## SVM 0.641358 0.8853519 0.9346716 0.9737669 1.037647 1.454771 0  
## MARS 0.911559 0.9945572 1.0656923 1.0832752 1.164013 1.322374 0  
##   
## RMSE   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## KNN 1.3153774 1.555852 1.651496 1.658807 1.749694 1.876011 0  
## SVM 0.8496792 1.133595 1.222412 1.265242 1.376083 1.764116 0  
## MARS 1.1350328 1.250174 1.353009 1.358553 1.451467 1.710016 0  
##   
## Rsquared   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## KNN 0.12978422 0.2040849 0.2375602 0.2440091 0.2935890 0.3425642 0  
## SVM 0.03787409 0.4937108 0.5928022 0.5595373 0.6750067 0.7242868 0  
## MARS 0.30190542 0.4604769 0.5379582 0.5138579 0.5813774 0.6426730 0

# Create performance function for testing   
ModTest <- function(x, testdf, testtarget) {  
 method <- c()  
 res <- data.frame()  
 for(x in x){  
 method <- c(method, x$method)  
 pred <- predict(x, newdata=testdf)  
 res <- rbind(res, t(postResample(pred=pred, obs=testtarget)))  
 }  
 row.names(res) <- method  
 return(res)  
}  
x <- list(mod.knn, mod.svm, mod.mar)  
ModTest.Results <- ModTest(x, testx, testy)  
ModTest.Results <- ModTest.Results %>%   
 mutate(Model = c("KNN", "SVM", "MARS")) %>%   
 dplyr::select(Model, RMSE, Rsquared, MAE)

#### Response

KNN, SVM, and MARS models were trained, summarized, tuned, and tested with the chemical data. The SVM model performs the best in its R squared value and has a lower MAE. For this we would use the SVM model. However, the MARS model produced a lower RMSE in our prediction test. See the model test results table based on predicted data for more descriptive statistics.

| **Model** | **RMSE** | **Rsquared** | **MAE** |
| --- | --- | --- | --- |
| KNN | 1.634191 | 0.1624915 | 1.2401878 |
| SVM | 1.115972 | 0.5943372 | 0.8672044 |
| MARS | 1.173855 | 0.5498068 | 0.9040343 |

### Part B

#### Question

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)  
imp.svm <- varImp(mod.svm)  
imp.mar <- varImp(mod.mar)  
impsvm.tbl <- imp.svm$importance %>%   
 as.data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,6)) %>%   
 dplyr::select(Rank, Predictor, Importance)  
impmar.tbl <- imp.mar$importance %>%   
 as.data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,6)) %>%   
 dplyr::select(Rank, Predictor, Importance)

#### Response

The process variables dominate the list regardless of how we review the models. The top ten most important predictors are majority process based. If we assume the SVM is our best and review its predictors we can produce the following rank for each predictor with its importance score:

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | ManufacturingProcess32 | 100.00000 |
| 2 | ManufacturingProcess13 | 89.32571 |
| 3 | BiologicalMaterial06 | 79.13916 |
| 4 | ManufacturingProcess17 | 73.38729 |
| 5 | BiologicalMaterial12 | 65.97102 |
| 6 | ManufacturingProcess09 | 64.97334 |
| 7 | ManufacturingProcess31 | 61.88981 |
| 8 | ManufacturingProcess36 | 61.33830 |
| 9 | BiologicalMaterial02 | 57.52609 |
| 10 | BiologicalMaterial03 | 57.22162 |

Alternatively, we can perform the same calculations on the MARS model if it were selected as our best and there would only be 3 predictors with ManufacturingProcess32 of most importance. Our MARS ranking would look shorter.

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | ManufacturingProcess32 | 100.00000 |
| 2 | ManufacturingProcess09 | 49.96277 |
| 3 | ManufacturingProcess13 | 0.00000 |

We note that the methods used to determine the optimal nonlinear regression model are different than those of the optimal linear model. For example, our varImp function in the caret package of R does not contain a method for selecting model type when evaluating predictor importance. Instead, it seems to filter the variables through regression, fit a LOESS line from predictor to outcome, and calculate an R squared term for each predictor. It then orders the predictors by R squared value in descending order, arranging the highest (and most important) predictors at the top. This is not the same for all model types.

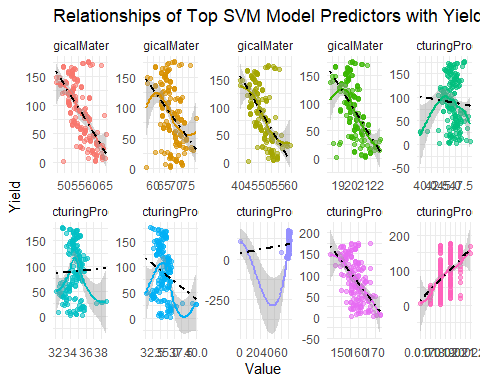
### Part C

#### Question

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(ggplot2)  
library(tidyr)  
impsvm.df <- impsvm.tbl  
df %>%   
 dplyr::select(c(impsvm.df$Predictor[1:10])) %>%   
 tibble::rownames\_to\_column("Yield.chr") %>%   
 mutate(Yield = as.numeric(Yield.chr)) %>%  
 dplyr::select(-Yield.chr) %>%   
 gather(key, value, -Yield) %>%   
 ggplot(aes(value, Yield, color=key)) +   
 geom\_point(aes(alpha = 0.25)) +  
 geom\_smooth() +   
 geom\_smooth(method = "lm",   
 linetype = "dotdash",   
 color = "black", se = F) +  
 theme(legend.position = "none") +   
 labs(x = "Value", y = "Yield", title = "Relationships of Top SVM Model Predictors with Yield") +   
 facet\_wrap(~key, scales = "free", ncol = 5) +   
 theme\_minimal() +   
 theme(legend.position = "none")



#### Response

Scatterplots with LOESS and linear estimates are fit to our 10 most important predictors from the SVM model to gain an understanding of their relationships with Yield as it is observed. We notice that ManufacturingProcess32 has the clearest linear relationship with yield and the process predictors each seem to have a single optimal value for maximizing yield (beyond which there are diminishing returns). Biological predictors remain less predictable to our model. Thus, we can say part of our intuition is confirmed based on our exploration of the data. However, our radial SVM model transformed the data such that we are not able to visualize its relationship clearly in two-dimensional space. Thus, the initial observations before modeling provide the best visualization to gain well-rounded understanding.

## KJ 8.1

8.1. Recreate the simulated data from Exercise 7.2:

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

### Part A

#### Question

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

#### Code

# Some packages used in later parts  
library(mlbench)  
library(randomForest)  
library(caret)  
library(partykit)  
library(dplyr)  
library(gbm)  
library(Cubist)  
library(flextable)  
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"  
model1 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp1 <- varImp(model1, scale = FALSE)  
rfImp1.tbl <- rfImp1 %>%  
 as.data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,3)) %>%   
 dplyr::select(Rank, Predictor, Importance)  
flextable(rfImp1.tbl) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | V1 | 8.732 |
| 2 | V4 | 7.615 |
| 3 | V2 | 6.415 |
| 4 | V5 | 2.024 |
| 5 | V3 | 0.764 |
| 6 | V6 | 0.165 |
| 7 | V7 | -0.006 |
| 8 | V10 | -0.075 |
| 9 | V9 | -0.095 |
| 10 | V8 | -0.166 |

#### Response

Given our ranked table of importance by predictor, V6 – V10 are near zero or negative indicating their insignificance in the random forest model.

### Part B

#### Question

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

#### Code

model2 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp2 <- varImp(model2, scale = FALSE)  
rfImp2.tbl <- rfImp2 %>%  
 as.data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,3)) %>%   
 dplyr::select(Rank, Predictor, Importance)  
flextable(rfImp2.tbl) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | V1 | 8.690 |
| 2 | V4 | 7.687 |
| 3 | V2 | 6.430 |
| 4 | V5 | 2.369 |
| 5 | V3 | 0.747 |
| 6 | V6 | 0.110 |
| 7 | V10 | 0.049 |
| 8 | V7 | 0.031 |
| 9 | V8 | -0.108 |
| 10 | V9 | -0.128 |

#### Response

A new table of ranked predictors by importance shows that when adding a predictor that is highly correlated with one of the informative predictors the importance of the predictor is split across the correlated variables. For example, adding a predictor highly correlated with V1 dropped its importance from about 9 to roughly 5 and the remaining importance was reassigned to the highly correlated ‘duplicate1’ variable. Predictors V4 and V2 held their importance the same. By maintaining the same level while V1’s importance dropped, both V4 and V2 became the two most importance predictors.

### Part C

#### Question

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)  
library(dplyr)  
cforest\_model <- cforest(y ~ ., data=simulated)  
# Unconditional importance measure  
varimp(cforest\_model) %>% sort(decreasing = T)

## V1 V4 V2 V5 V3 V7   
## 7.91634213 7.53062366 6.16317586 1.99099762 0.27681088 0.16245729   
## V6 V10 V8 V9   
## 0.06420066 -0.03584637 -0.06861244 -0.11560311

varimp.tbl <- varimp(cforest\_model, conditional=T) %>% sort(decreasing = T)  
varimp.tbl <- varimp.tbl %>%  
 as.data.frame() %>%  
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(.)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(.,3)) %>%   
 dplyr::select(Rank, Predictor, Importance)  
flextable(varimp.tbl) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | V4 | 6.250 |
| 2 | V1 | 6.234 |
| 3 | V2 | 5.391 |
| 4 | V5 | 1.407 |
| 5 | V3 | 0.141 |
| 6 | V7 | -0.035 |
| 7 | V6 | -0.187 |
| 8 | V10 | -0.306 |
| 9 | V8 | -0.352 |
| 10 | V9 | -0.368 |

#### Response

The uninformative predictors V6 - V10 are still rated with low importance. Predictor ‘duplicate1’ (that is highly correlated with V1) was reduced in importance from the varimp() function. All other importance values are also reduced. From this, we can conclude the patterns between traditional and modified measures of importance are very similar in order and arrangement, but the modified measure reduces predictor importance to a smaller scale. When importance is already small, this may reorder the insignificant predictors as V6-V10 have been.

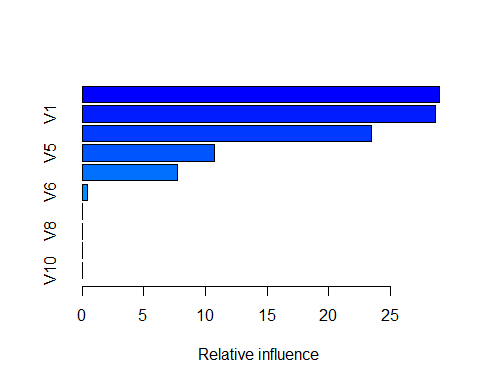
### Part D

#### Question

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

#### Code

library(gbm)  
library(Cubist)  
gbm\_Model <- gbm(y ~ ., data=simulated, distribution='gaussian')  
gbm.summary <- summary(gbm\_Model)  
gbm.tbl <- gbm.summary %>%   
 as.data.frame() %>%   
 arrange(desc(rel.inf)) %>%   
 mutate(Rank = row\_number(),   
 Predictor = var,  
 Importance = round(rel.inf,3)) %>%   
 dplyr::select(Rank, Predictor, Importance)  
summary(gbm\_Model)



## var rel.inf  
## V4 V4 29.0111902  
## V1 V1 28.6900508  
## V2 V2 23.4378987  
## V5 V5 10.7687292  
## V3 V3 7.6935092  
## V6 V6 0.3986219  
## V7 V7 0.0000000  
## V8 V8 0.0000000  
## V9 V9 0.0000000  
## V10 V10 0.0000000

cubistModel <- cubist(x=simulated[,-(ncol(simulated)-1)], y=simulated$y, committees=100)  
cube.varimp.tbl <- varImp(cubistModel) %>%  
 as.data.frame() %>%  
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,3)) %>%   
 dplyr::select(Rank, Predictor, Importance)

#### Response

The top-rated predictor is V4 in a GBM model and V1 in a Cubist model. Predictors V6 - V10 remain very low in importance. The predictor ‘duplicate1’ scored a 0 for importance in the Cubist model but higher in GBM. We review the patterns through two tables with the predictors from GBM first.

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | V4 | 29.011 |
| 2 | V1 | 28.690 |
| 3 | V2 | 23.438 |
| 4 | V5 | 10.769 |
| 5 | V3 | 7.694 |
| 6 | V6 | 0.399 |
| 7 | V7 | 0.000 |
| 8 | V8 | 0.000 |
| 9 | V9 | 0.000 |
| 10 | V10 | 0.000 |

For GBM compared to random forest, the scale of importance to the model has changed. In GBM no value is negative. The lowest bound is 0. GBM is also a measure of relative influence, not strictly importance which seems to follow a more exponential scale from most to least important predictors. We also review the Cubist model’s predictors.

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | y | 50 |
| 2 | V1 | 0 |
| 3 | V2 | 0 |
| 4 | V3 | 0 |
| 5 | V4 | 0 |
| 6 | V5 | 0 |
| 7 | V6 | 0 |
| 8 | V7 | 0 |
| 9 | V8 | 0 |
| 10 | V9 | 0 |

For Cubist model the change in vector magnitude from least to most important is even greater than GBM (and far from any random forest). It is also bounded by 0 at its lowest like GBM but the orders are slightly different. Predictor ‘duplicate1’ is ranked 5th in both models but V1 ranks higher in Cubist.

Based on the observations above, the pattern is different across the models.

## KJ 8.2

Use a simulation to show tree bias with different granularities.

### Part A

#### Question

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

#### Code

library(caret)  
library(rpart)  
set.seed(755)  
X1 <- rep(1:2, each=100)  
Y <- X1 + rnorm(600, mean=2, sd=4)  
X2 <- rnorm(600, mean=2, sd=4)  
simData <- data.frame(Y=Y, X1=X1, X2=X2)  
fit <- rpart(Y ~ ., data = simData)  
v1 <- round(var(X1), 3)  
v2 <- round(var(X2), 3)  
sim.tbl1 <- varImp(fit) %>%   
 as.data.frame() %>%   
 tibble::rownames\_to\_column("Simulation") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 RepQuantity = 600,  
 Variance = c(v1, v2),  
 Importance = round(Overall,3)) %>%   
 dplyr::select(Rank, Simulation, RepQuantity, Variance, Importance)  
flextable(sim.tbl1) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Rank** | **Simulation** | **RepQuantity** | **Variance** | **Importance** |
| --- | --- | --- | --- | --- |
| 1 | X1 | 600 | 0.251 | 0.022 |
| 2 | X2 | 600 | 14.212 | 0.005 |

#### Response

The simulation was created with two variables. Each was assigned a mean of 2 and standard deviation of 4. The two variables differ in variance.

\* X1 -> Lesser variance (Sequence of values)   
\* X2 -> Higher Variance (Gaussian Random value)  
\* Target Variable Y -> X1 + X2

A regression tree is fitted using recursive partitioning and importance calculated with the caret package’s varImp() method. We can see simulation ‘X1’ has greater importance than ‘X2.’ The variance also differs with ‘X2’ being over 56 times greater than ‘X1.’ The difference between two variables is significant and it shows the tree bias between two variables.

## KJ 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:

[insert figure 8.24]

### Part A

#### Question

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

There could be two reasons the model on the right focuses its importance on the first few predictors while the model on the left spreads its importance out. These are outlined below:

**Bragging Fraction** – it represent the data usage in each interation of trees. The plot on the left has a bragging fraction of 0.1 which is low and only 105 of the data is used for random sampling. Whereas, the plot on the right has a bragging fraction of 0.9 which is quite large comparatively. It indicates 90% of the data is used on each iteration which is alsmot the full data set. Since almost the full data set is used (by the model on the right), a handful of predictors are able to pick up more importance. This concentrates the importance as shown in the plot on the right. The opposite occurs in the plot on the left with where only 10% of the data set is used.

**Learning Rate** – it means a higher number of predictions are added to the model output. Since the plot on the right has more predictions, the correlation is greater. Thus, only the first few predictors were considered significant.

### Part B

#### Question

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

#### Response

Since the bragging fraction and learning rate can control overfitting of the model, they are crucial in selecting the more predictive model. If we are trying to cover more predictions for unseen samples, then a smaller bragging fraction and lower learning rate should be more predictive than one with a larger fraction and higher rate. This is because it includes a higher proportion of possible sample predictions. Thus, the left plot would probably be more predictive of other samples. However, there is a point that one must make a bias-variance compromise to remain usefully predictive.

### Part C

#### Question

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

#### Code

library(gbm)  
library(AppliedPredictiveModeling)  
data(solubility)  
grid1 <- expand.grid(n.trees=100, interaction.depth=1, shrinkage=0.1, n.minobsinnode=10)  
gbm1 <- train(x = solTrainXtrans, y = solTrainY, method = 'gbm', tuneGrid = grid1, verbose = FALSE)  
grid2 <- expand.grid(n.trees=100, interaction.depth=10, shrinkage=0.1, n.minobsinnode=10)  
gbm2 <- train(x = solTrainXtrans, y = solTrainY, method = 'gbm', tuneGrid = grid2, verbose = FALSE)  
gbm1.df <- varImp(gbm1)  
gbm2.df <- varImp(gbm2)  
gbm1.tbl <- gbm1.df$importance %>%   
 as.data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,3)) %>%   
 dplyr::select(Rank, Predictor, Importance)  
gbm2.tbl <- gbm2.df$importance %>%   
 as.data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Rank = row\_number(),   
 Importance = round(Overall,3)) %>%   
 dplyr::select(Rank, Predictor, Importance)

#### Response

Increasing interaction depth improves predictor importance significantly. Two tables were created from two GBM models measuring importance of each predictor. They show the first 10 predictors and their rank by importance in descending order. The only difference between the models is interaction depth. The first contains an interaction depth of 1 and is shown:

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | NumCarbon | 100.000 |
| 2 | MolWeight | 63.707 |
| 3 | SurfaceArea2 | 36.382 |
| 4 | NumAromaticBonds | 24.449 |
| 5 | NumChlorine | 19.016 |
| 6 | SurfaceArea1 | 16.448 |
| 7 | HydrophilicFactor | 11.640 |
| 8 | NumHalogen | 6.092 |
| 9 | NumMultBonds | 6.049 |
| 10 | FP112 | 4.867 |

The second contains an interaction depth of 10. Notice the difference in importance scores and ranked order of importance compared to the first table.

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | NumCarbon | 100.000 |
| 2 | MolWeight | 50.699 |
| 3 | SurfaceArea1 | 32.876 |
| 4 | SurfaceArea2 | 24.533 |
| 5 | HydrophilicFactor | 16.340 |
| 6 | NumNonHAtoms | 10.719 |
| 7 | NumChlorine | 6.765 |
| 8 | NumMultBonds | 5.529 |
| 9 | NumAromaticBonds | 5.165 |
| 10 | NumHydrogen | 4.443 |

The order of predictors changed (see ranks 3 – 10) and the importance decreased quicker with greater interaction depth. With the interaction depth at 10, the importance spread across more predictors shrinking the importance values for the first 10 predictors in the second table.

## KJ 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:

### Part A

#### Question

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

#### Code

# Code from exercise 6.3 and 7.5  
library(AppliedPredictiveModeling)  
library(Metrics)  
library(reshape2)  
library(dplyr)  
library(caret)  
set.seed(0130)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
sum(is.na(train))

## [1] 0

sum(is.na(test))

## [1] 0

# Create the Random Forest Tree  
rf\_model = randomForest(Yield~.,   
 data=train)

# Apply 6.3 and 7.5   
# Predict with the test data  
set.seed(0130)  
rf\_test = predict(rf\_model, newdata = test)  
rf\_test\_df = data.frame(rf\_test)  
rf\_test\_df$actual = test$Yield  
rf\_test\_rmse = rmse(rf\_test\_df$actual, rf\_test\_df$rf\_test)  
# Random forest model  
rfMod <- train(x = trainx,  
 y = trainy,  
 method = 'rf',  
 tuneLength = 10)  
#Create the ctree   
ctree\_model = cforest(Yield~.,   
 data=train)  
#Predict with the test data  
ctree\_test = predict(ctree\_model, newdata = test)  
ctree\_test\_df = data.frame(ctree\_test)  
ctree\_test\_df$actual = test$Yield  
# Uses lmyield from 6.3   
# Recreate lmyield and RMSE from 6.3   
# Part A load the data  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
# Part B Fill in missing values with the median of each feature  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}  
# Part C Correlations and Modeling  
cormat <- round(cor(df),2)  
melted\_cormat = melt(cormat)  
melted\_cormat\_df = data.frame(melted\_cormat)  
#Filter to only Yield and sort in descending order  
yield\_corr <- melted\_cormat\_df %>%   
 filter(Var2 == "Yield")  
yield\_corr$absvalue = abs(yield\_corr$value)  
yield\_corr2 = yield\_corr[order(-yield\_corr[,4]),]  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
#Create the regression model  
lmyield = lm(Yield~.,   
 data=train)  
# Part D Predictions  
# Predict with the test data  
lmyield\_test = predict(lmyield, newdata = test, interval ='prediction')  
lmyield\_test\_df = data.frame(lmyield\_test)  
lmyield\_test\_df$actual = test$Yield  
lmyield\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
# Create ctree include lmyield  
ctree\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
ctreeMod <- train(x = trainx,  
 y = trainy,  
 method = 'ctree2',  
 tuneLength = 10)  
# cForest Model  
cforestMod <- train(x = trainx,  
 y = trainy,  
 method = 'cforest',  
 tuneLength = 10)  
# Single tree   
singtree <- train(x = trainx,  
 y = trainy,  
 method = "rpart",  
 tuneLength = 10,  
 control = rpart.control(maxdepth=2))  
# Cubist model  
cubeMod <- train(x = trainx,  
 y = trainy,  
 method = 'cubist')  
# Tabulate performance statistics  
ct.res <- postResample(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
rf.res <- postResample(rf\_test\_df$actual, rf\_test\_df$rf\_test)  
res <- resamples(list(SingleTree = singtree,   
 RandomForest = rfMod,  
 CTree = ctreeMod,   
 Cubist = cubeMod,   
 CForest = cforestMod))  
x <- list(singtree,   
 rfMod,   
 ctreeMod,   
 cubeMod,   
 cforestMod)  
ModTest.Results <- ModTest(x, testx, testy)  
ModTest.Results <- ModTest.Results %>%   
 mutate(Model = c("SingleTree", "RandomForest", "CTree", "Cubist", "CForest")) %>%   
 dplyr::select(Model, RMSE, Rsquared, MAE)  
flextable(ModTest.Results) %>%   
 theme\_vanilla() %>%   
 set\_table\_properties(layout = "autofit")

| **Model** | **RMSE** | **Rsquared** | **MAE** |
| --- | --- | --- | --- |
| SingleTree | 1.298081 | 0.4587047 | 1.0360354 |
| RandomForest | 1.039834 | 0.6517535 | 0.8036644 |
| CTree | 1.411750 | 0.3460955 | 1.1460436 |
| Cubist | 1.022040 | 0.6708036 | 0.8231436 |
| CForest | 1.151959 | 0.5643823 | 0.8909904 |

#### Response

Comparing SingleTree, RandomForest, CTree, Cubist, and CForest models, we found that the cubist model performed best due to lower root mean squared error RMSE, higher coefficient of determination (Rsquared), and lowest mean absolute error MAE. Although this was a single evaluation. Future tests could switch these results as we would expect with prediction of unknown data sources.

Additionally, determining the optimal model often depends on its application and which source of error you attempt to reduce the most. For prediction in general, a lower RMSE would probably fare better than Rsquared and take priority over MAE but it all depends on the problem and circumstances.

### Part B

#### Question

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

rf <- varImp(rf\_model)  
ct <- varimp(ctree\_model)  
rf <- rf %>%   
 data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(Overall)) %>%   
 mutate(Importance = round(Overall, 3),   
 Rank = row\_number()) %>%  
 dplyr::select(Rank, Predictor, Importance)   
ct <- ct %>%   
 data.frame() %>%   
 tibble::rownames\_to\_column("Predictor") %>%   
 arrange(desc(.)) %>%   
 mutate(Importance = round(., 3),   
 Rank = row\_number()) %>%  
 dplyr::select(Rank, Predictor, Importance)   
rpart

## function (formula, data, weights, subset, na.action = na.rpart,   
## method, model = FALSE, x = FALSE, y = TRUE, parms, control,   
## cost, ...)   
## {  
## Call <- match.call()  
## if (is.data.frame(model)) {  
## m <- model  
## model <- FALSE  
## }  
## else {  
## indx <- match(c("formula", "data", "weights", "subset"),   
## names(Call), nomatch = 0L)  
## if (indx[1] == 0L)   
## stop("a 'formula' argument is required")  
## temp <- Call[c(1L, indx)]  
## temp$na.action <- na.action  
## temp[[1L]] <- quote(stats::model.frame)  
## m <- eval.parent(temp)  
## }  
## Terms <- attr(m, "terms")  
## if (any(attr(Terms, "order") > 1L))   
## stop("Trees cannot handle interaction terms")  
## Y <- model.response(m)  
## wt <- model.weights(m)  
## if (any(wt < 0))   
## stop("negative weights not allowed")  
## if (!length(wt))   
## wt <- rep(1, nrow(m))  
## offset <- model.offset(m)  
## X <- rpart.matrix(m)  
## nobs <- nrow(X)  
## nvar <- ncol(X)  
## if (missing(method)) {  
## method <- if (is.factor(Y) || is.character(Y))   
## "class"  
## else if (inherits(Y, "Surv"))   
## "exp"  
## else if (is.matrix(Y))   
## "poisson"  
## else "anova"  
## }  
## if (is.list(method)) {  
## mlist <- method  
## method <- "user"  
## init <- if (missing(parms))   
## mlist$init(Y, offset, wt = wt)  
## else mlist$init(Y, offset, parms, wt)  
## keep <- rpartcallback(mlist, nobs, init)  
## method.int <- 4L  
## parms <- init$parms  
## }  
## else {  
## method.int <- pmatch(method, c("anova", "poisson", "class",   
## "exp"))  
## if (is.na(method.int))   
## stop("Invalid method")  
## method <- c("anova", "poisson", "class", "exp")[method.int]  
## if (method.int == 4L)   
## method.int <- 2L  
## init <- if (missing(parms))   
## get(paste("rpart", method, sep = "."), envir = environment())(Y,   
## offset, , wt)  
## else get(paste("rpart", method, sep = "."), envir = environment())(Y,   
## offset, parms, wt)  
## ns <- asNamespace("rpart")  
## if (!is.null(init$print))   
## environment(init$print) <- ns  
## if (!is.null(init$summary))   
## environment(init$summary) <- ns  
## if (!is.null(init$text))   
## environment(init$text) <- ns  
## }  
## Y <- init$y  
## xlevels <- .getXlevels(Terms, m)  
## cats <- rep(0L, ncol(X))  
## if (!is.null(xlevels))   
## cats[match(names(xlevels), colnames(X))] <- unlist(lapply(xlevels,   
## length))  
## extraArgs <- list(...)  
## if (length(extraArgs)) {  
## controlargs <- names(formals(rpart.control))  
## indx <- match(names(extraArgs), controlargs, nomatch = 0L)  
## if (any(indx == 0L))   
## stop(gettextf("Argument %s not matched", names(extraArgs)[indx ==   
## 0L]), domain = NA)  
## }  
## controls <- rpart.control(...)  
## if (!missing(control))   
## controls[names(control)] <- control  
## xval <- controls$xval  
## if (is.null(xval) || (length(xval) == 1L && xval == 0L) ||   
## method == "user") {  
## xgroups <- 0L  
## xval <- 0L  
## }  
## else if (length(xval) == 1L) {  
## xgroups <- sample(rep(1L:xval, length = nobs), nobs,   
## replace = FALSE)  
## }  
## else if (length(xval) == nobs) {  
## xgroups <- xval  
## xval <- length(unique(xgroups))  
## }  
## else {  
## if (!is.null(attr(m, "na.action"))) {  
## temp <- as.integer(attr(m, "na.action"))  
## xval <- xval[-temp]  
## if (length(xval) == nobs) {  
## xgroups <- xval  
## xval <- length(unique(xgroups))  
## }  
## else stop("Wrong length for 'xval'")  
## }  
## else stop("Wrong length for 'xval'")  
## }  
## if (missing(cost))   
## cost <- rep(1, nvar)  
## else {  
## if (length(cost) != nvar)   
## stop("Cost vector is the wrong length")  
## if (any(cost <= 0))   
## stop("Cost vector must be positive")  
## }  
## tfun <- function(x) if (is.matrix(x))   
## rep(is.ordered(x), ncol(x))  
## else is.ordered(x)  
## labs <- sub("^`(.\*)`$", "\\1", attr(Terms, "term.labels"))  
## isord <- unlist(lapply(m[labs], tfun))  
## storage.mode(X) <- "double"  
## storage.mode(wt) <- "double"  
## temp <- as.double(unlist(init$parms))  
## if (!length(temp))   
## temp <- 0  
## rpfit <- .Call(C\_rpart, ncat = as.integer(cats \* !isord),   
## method = as.integer(method.int), as.double(unlist(controls)),   
## temp, as.integer(xval), as.integer(xgroups), as.double(t(init$y)),   
## X, wt, as.integer(init$numy), as.double(cost))  
## nsplit <- nrow(rpfit$isplit)  
## ncat <- if (!is.null(rpfit$csplit))   
## nrow(rpfit$csplit)  
## else 0L  
## if (nsplit == 0L)   
## xval <- 0L  
## numcp <- ncol(rpfit$cptable)  
## temp <- if (nrow(rpfit$cptable) == 3L)   
## c("CP", "nsplit", "rel error")  
## else c("CP", "nsplit", "rel error", "xerror", "xstd")  
## dimnames(rpfit$cptable) <- list(temp, 1L:numcp)  
## tname <- c("<leaf>", colnames(X))  
## splits <- matrix(c(rpfit$isplit[, 2:3], rpfit$dsplit), ncol = 5L,   
## dimnames = list(tname[rpfit$isplit[, 1L] + 1L], c("count",   
## "ncat", "improve", "index", "adj")))  
## index <- rpfit$inode[, 2L]  
## nadd <- sum(isord[rpfit$isplit[, 1L]])  
## if (nadd > 0L) {  
## newc <- matrix(0L, nadd, max(cats))  
## cvar <- rpfit$isplit[, 1L]  
## indx <- isord[cvar]  
## cdir <- splits[indx, 2L]  
## ccut <- floor(splits[indx, 4L])  
## splits[indx, 2L] <- cats[cvar[indx]]  
## splits[indx, 4L] <- ncat + 1L:nadd  
## for (i in 1L:nadd) {  
## newc[i, 1L:(cats[(cvar[indx])[i]])] <- -as.integer(cdir[i])  
## newc[i, 1L:ccut[i]] <- as.integer(cdir[i])  
## }  
## catmat <- if (ncat == 0L)   
## newc  
## else {  
## cs <- rpfit$csplit  
## ncs <- ncol(cs)  
## ncc <- ncol(newc)  
## if (ncs < ncc)   
## cs <- cbind(cs, matrix(0L, nrow(cs), ncc - ncs))  
## rbind(cs, newc)  
## }  
## ncat <- ncat + nadd  
## }  
## else catmat <- rpfit$csplit  
## if (nsplit == 0L) {  
## frame <- data.frame(row.names = 1L, var = "<leaf>", n = rpfit$inode[,   
## 5L], wt = rpfit$dnode[, 3L], dev = rpfit$dnode[,   
## 1L], yval = rpfit$dnode[, 4L], complexity = rpfit$dnode[,   
## 2L], ncompete = 0L, nsurrogate = 0L)  
## }  
## else {  
## temp <- ifelse(index == 0L, 1L, index)  
## svar <- ifelse(index == 0L, 0L, rpfit$isplit[temp, 1L])  
## frame <- data.frame(row.names = rpfit$inode[, 1L], var = tname[svar +   
## 1L], n = rpfit$inode[, 5L], wt = rpfit$dnode[, 3L],   
## dev = rpfit$dnode[, 1L], yval = rpfit$dnode[, 4L],   
## complexity = rpfit$dnode[, 2L], ncompete = pmax(0L,   
## rpfit$inode[, 3L] - 1L), nsurrogate = rpfit$inode[,   
## 4L])  
## }  
## if (method.int == 3L) {  
## numclass <- init$numresp - 2L  
## nodeprob <- rpfit$dnode[, numclass + 5L]/sum(wt)  
## temp <- pmax(1L, init$counts)  
## temp <- rpfit$dnode[, 4L + (1L:numclass)] %\*% diag(init$parms$prior/temp)  
## yprob <- temp/rowSums(temp)  
## yval2 <- matrix(rpfit$dnode[, 4L + (0L:numclass)], ncol = numclass +   
## 1L)  
## frame$yval2 <- cbind(yval2, yprob, nodeprob)  
## }  
## else if (init$numresp > 1L)   
## frame$yval2 <- rpfit$dnode[, -(1L:3L), drop = FALSE]  
## if (is.null(init$summary))   
## stop("Initialization routine is missing the 'summary' function")  
## functions <- if (is.null(init$print))   
## list(summary = init$summary)  
## else list(summary = init$summary, print = init$print)  
## if (!is.null(init$text))   
## functions <- c(functions, list(text = init$text))  
## if (method == "user")   
## functions <- c(functions, mlist)  
## where <- rpfit$which  
## names(where) <- row.names(m)  
## ans <- list(frame = frame, where = where, call = Call, terms = Terms,   
## cptable = t(rpfit$cptable), method = method, parms = init$parms,   
## control = controls, functions = functions, numresp = init$numresp)  
## if (nsplit)   
## ans$splits = splits  
## if (ncat > 0L)   
## ans$csplit <- catmat + 2L  
## if (nsplit)   
## ans$variable.importance <- importance(ans)  
## if (model) {  
## ans$model <- m  
## if (missing(y))   
## y <- FALSE  
## }  
## if (y)   
## ans$y <- Y  
## if (x) {  
## ans$x <- X  
## ans$wt <- wt  
## }  
## ans$ordered <- isord  
## if (!is.null(attr(m, "na.action")))   
## ans$na.action <- attr(m, "na.action")  
## if (!is.null(xlevels))   
## attr(ans, "xlevels") <- xlevels  
## if (method == "class")   
## attr(ans, "ylevels") <- init$ylevels  
## class(ans) <- "rpart"  
## ans  
## }  
## <bytecode: 0x0000000029e5b9f8>  
## <environment: namespace:rpart>

#### Response

The most important predictors in every model run thus far, including these two tree-based models, have been process predictors. The process variables held the majority of spots in the top 10 most important predictors in almost every model run thus far as well. However, some biological predictors creeped into the top 10 with our ctree and random forest models. Take a look at the ranked 2, 3, 5, and 9 spots in the table of importance from the random forest model:

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | ManufacturingProcess32 | 1.202 |
| 2 | BiologicalMaterial03 | 0.911 |
| 3 | BiologicalMaterial06 | 0.775 |
| 4 | BiologicalMaterial12 | 0.679 |
| 5 | ManufacturingProcess13 | 0.649 |
| 6 | ManufacturingProcess09 | 0.593 |
| 7 | ManufacturingProcess36 | 0.587 |
| 8 | BiologicalMaterial02 | 0.557 |
| 9 | ManufacturingProcess12 | 0.513 |
| 10 | BiologicalMaterial04 | 0.457 |

Now, look at ranked spots 2, 3, 5, 9, and 10 in our ctree model. Importance values have also changed scale, maxing out near 100, while our ctree maximum importance value was just over 1.

| **Rank** | **Predictor** | **Importance** |
| --- | --- | --- |
| 1 | ManufacturingProcess32 | 1.202 |
| 2 | BiologicalMaterial03 | 0.911 |
| 3 | BiologicalMaterial06 | 0.775 |
| 4 | BiologicalMaterial12 | 0.679 |
| 5 | ManufacturingProcess13 | 0.649 |
| 6 | ManufacturingProcess09 | 0.593 |
| 7 | ManufacturingProcess36 | 0.587 |
| 8 | BiologicalMaterial02 | 0.557 |
| 9 | ManufacturingProcess12 | 0.513 |
| 10 | BiologicalMaterial04 | 0.457 |

In any case, the top 10 predictors in linear and nonlinear models imply the same general rule, that manufacturing processes (especially ManufacturingProcess32, ManufacturingProcess13, and ManufacturingProcess09) are the most important predictors of Yield. If one desires to maximize Yield, it would be best advised to improve these processes.

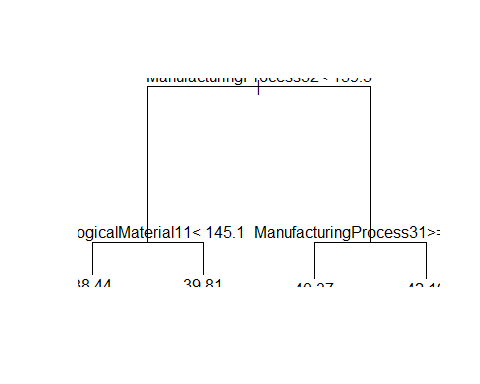
### Part C

#### Question

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?

#### Code

plot(singtree$finalModel)  
text(singtree$finalModel)



#### Response

Through the hierarchical structure shown in this plot of a single tree model, we can easily determine the most important predictor (ManufacturingProcess32) and its relationship with Yield through nodes (or limbs since we are using tree references). It also shows us how importance it gets distributed to the next two most important predictors in this data set. We could continue this through all the branches of a single model to see where all predictors are ranked.

For example, the single split in this model uses data from ManufacturingProcess32, which we suspected was the most significant influencer of Yield from our previous models. From this view we also notice that Yield and ManufacturingProcess32 are linearly related. An increase in one should result in an increase in the other. Since this is single tree model, we know that values are assigned by computing the mean of the group and splitting it by our single most important predictor, which is almost certainly, ManufacturingProcess32.

## Market Basket Analysis

#### Question

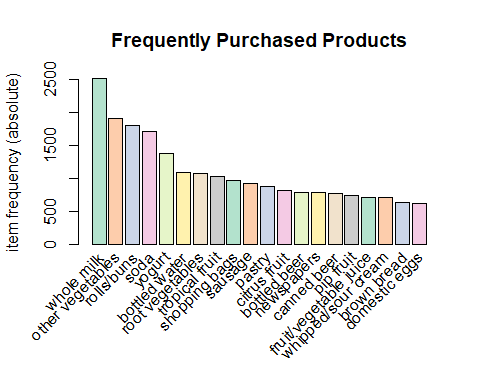
I am assigning one simple problem on market basket analysis / recommender systems. Imagine 10000 receipts sitting on your table. Each receipt represents a transaction with items that were purchased. The receipt is a representation of stuff that went into a customer’s basket – and therefore ‘Market Basket Analysis’. That is exactly what the Groceries Data Set contains: a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a transaction and each column in a row represents an item. Here is the dataset = GroceryDataSet.csv (comma separated file) Your 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)  
library(arulesViz)  
library(RColorBrewer)  
library(visNetwork)  
library(igraph)  
df <- read.csv(  
"https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/GroceryDataSet.csv")  
df\_sparse <- read.transactions(  
"https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/GroceryDataSet.csv",  
 format="basket",sep=",")  
summary(df\_sparse)

## 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 16   
## 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55 46   
## 17 18 19 20 21 22 23 24 26 27 28 29 32   
## 29 14 14 9 11 4 6 1 1 1 1 3 1   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 2.000 3.000 4.409 6.000 32.000   
##   
## includes extended item information - examples:  
## labels  
## 1 abrasive cleaner  
## 2 artif. sweetener  
## 3 baby cosmetics

itemFrequencyPlot(df\_sparse,topN=20,type="absolute",col=brewer.pal(8,'Pastel2'), main="Frequently Purchased Products")



association.rules <- apriori(df\_sparse, parameter = list(supp=0.004, conf=0.3))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.3 0.1 1 none FALSE TRUE 5 0.004 1  
## maxlen target ext  
## 10 rules TRUE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 39   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].  
## sorting and recoding items ... [126 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 5 done [0.00s].  
## writing ... [735 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

length(association.rules)

## [1] 735

inspect(sort(association.rules, by = 'lift')[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {liquor} => {bottled beer} 0.004677173 0.4220183 0.011082867 5.240594 46  
## [2] {berries,   
## whole milk} => {whipped/sour cream} 0.004270463 0.3620690 0.011794611 5.050990 42  
## [3] {herbs,   
## whole milk} => {root vegetables} 0.004168785 0.5394737 0.007727504 4.949369 41  
## [4] {citrus fruit,   
## other vegetables,   
## tropical fruit} => {root vegetables} 0.004473818 0.4943820 0.009049314 4.535678 44  
## [5] {other vegetables,   
## root vegetables,   
## tropical fruit} => {citrus fruit} 0.004473818 0.3636364 0.012302999 4.393567 44  
## [6] {citrus fruit,   
## other vegetables,   
## root vegetables} => {tropical fruit} 0.004473818 0.4313725 0.010371124 4.110997 44  
## [7] {citrus fruit,   
## other vegetables,   
## whole milk} => {root vegetables} 0.005795628 0.4453125 0.013014743 4.085493 57  
## [8] {citrus fruit,   
## root vegetables,   
## tropical fruit} => {other vegetables} 0.004473818 0.7857143 0.005693950 4.060694 44  
## [9] {herbs} => {root vegetables} 0.007015760 0.4312500 0.016268429 3.956477 69  
## [10] {tropical fruit,   
## whipped/sour cream,   
## whole milk} => {yogurt} 0.004372140 0.5512821 0.007930859 3.951792 43

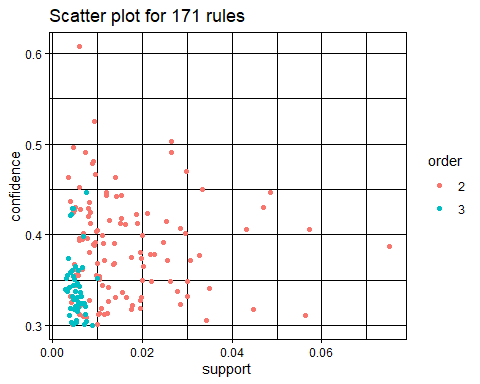
subset.rules <- which(colSums(is.subset(association.rules, association.rules)) > 1) # get subset rules in vector  
length(subset.rules)

## [1] 564

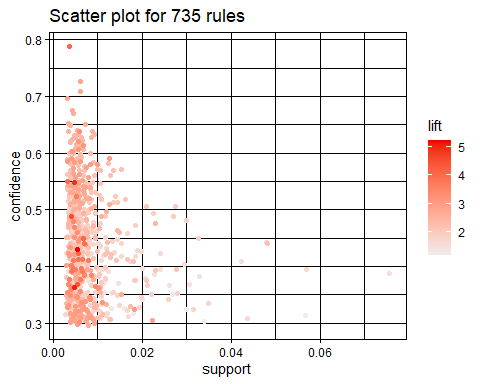
subset.association.rules. <- association.rules[-subset.rules] # remove subset rules.  
inspect(sort(subset.association.rules., by = 'lift')[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {liquor} => {bottled beer} 0.004677173 0.4220183 0.011082867 5.240594 46  
## [2] {herbs} => {root vegetables} 0.007015760 0.4312500 0.016268429 3.956477 69  
## [3] {citrus fruit,   
## pip fruit} => {tropical fruit} 0.005592272 0.4044118 0.013828165 3.854060 55  
## [4] {pip fruit,   
## root vegetables} => {tropical fruit} 0.005287239 0.3398693 0.015556685 3.238967 52  
## [5] {tropical fruit,   
## whipped/sour cream} => {yogurt} 0.006202339 0.4485294 0.013828165 3.215224 61  
## [6] {margarine,   
## tropical fruit} => {yogurt} 0.004067107 0.4347826 0.009354347 3.116681 40  
## [7] {citrus fruit,   
## root vegetables} => {tropical fruit} 0.005693950 0.3218391 0.017691917 3.067139 56  
## [8] {pip fruit,   
## rolls/buns} => {tropical fruit} 0.004473818 0.3211679 0.013929842 3.060742 44  
## [9] {beef} => {root vegetables} 0.017386884 0.3313953 0.052465684 3.040367 171  
## [10] {tropical fruit,   
## whipped/sour cream} => {root vegetables} 0.004575496 0.3308824 0.013828165 3.035660 45

plot(subset.association.rules.,method="two-key plot")

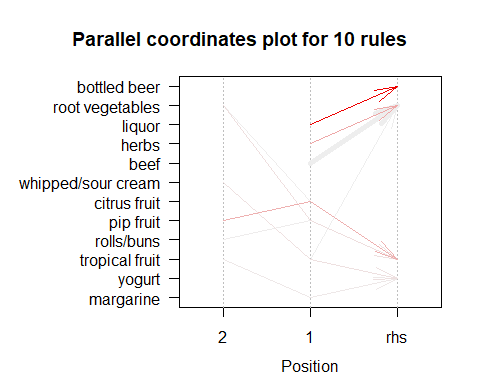


plot(association.rules)



top10subRules <- head(subset.association.rules., n = 10, by = "lift")  
plot(top10subRules, method = "graph", engine = "htmlwidget")

subRules2<-head(subset.association.rules., n=10, by="lift")  
plot(subRules2, method="paracoord")



#### Response

Our association rules mining was achieved using apriori algorithm functions available in the arules package, which we use throughout this analysis. Association rules mining is a two-step process:

1. Frequent Itemset Generation   
2. Rules Generation

Once the rules are generated, the top 10 rules were classified based on the output. To get there we loaded and summarized the data set as a sparce matrix. We then found the most frequently purchased products. The top 20 are shown in our code. We created our choice of minimum support to help classify the data. This formula is:

(Products purchased at least 5 times a day)/total number of transactions Supp = (6\*7)/nrow(df) = 0.004

Through trial and error on different combinations of items, we found that the confidence of 0.3 generated a decent number of rules for our combination. This produced a total of 735 rules and the top 10 rules are displayed as our “Parallel coordinates plot for 10 rules.” It appears the highest association is created for liquor and bottled beer with the life value of 5.24.

Going further, we cleaned up the redundant rules by subseting and down selecting our combination. The resultant highest association did not change but the other rules did. The results are shown in the “Parallel coordinates plot for 10 rules.” The most dependent purchases remained liquor and bottled beer with the life value of 5.24.