Data 624_Exercise_HW8

Jiaxin Zheng

2025-04-12

Kuhn and Johnson, Chapter 8

Exercise 7.2

```
library(tidyverse)
```

Friedman (1991) introduced several benchmark data sets create by simulation. The package mlbench contains a function called mlbench.friedman1 that simulates these data,

```
## Warning: package 'ggplot2' was built under R version 4.4.2
```

```
## Warning: package 'readr' was built under R version 4.4.3
```

```
## Warning: package 'dplyr' was built under R version 4.4.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 4.4.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.3
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(earth)
```

```
## Warning: package 'earth' was built under R version 4.4.3
```

```
## Loading required package: Formula
## Loading required package: plotmo
```

```
## Warning: package 'plotmo' was built under R version 4.4.3
```

```
## Loading required package: plotrix
```

```
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.4.2
```

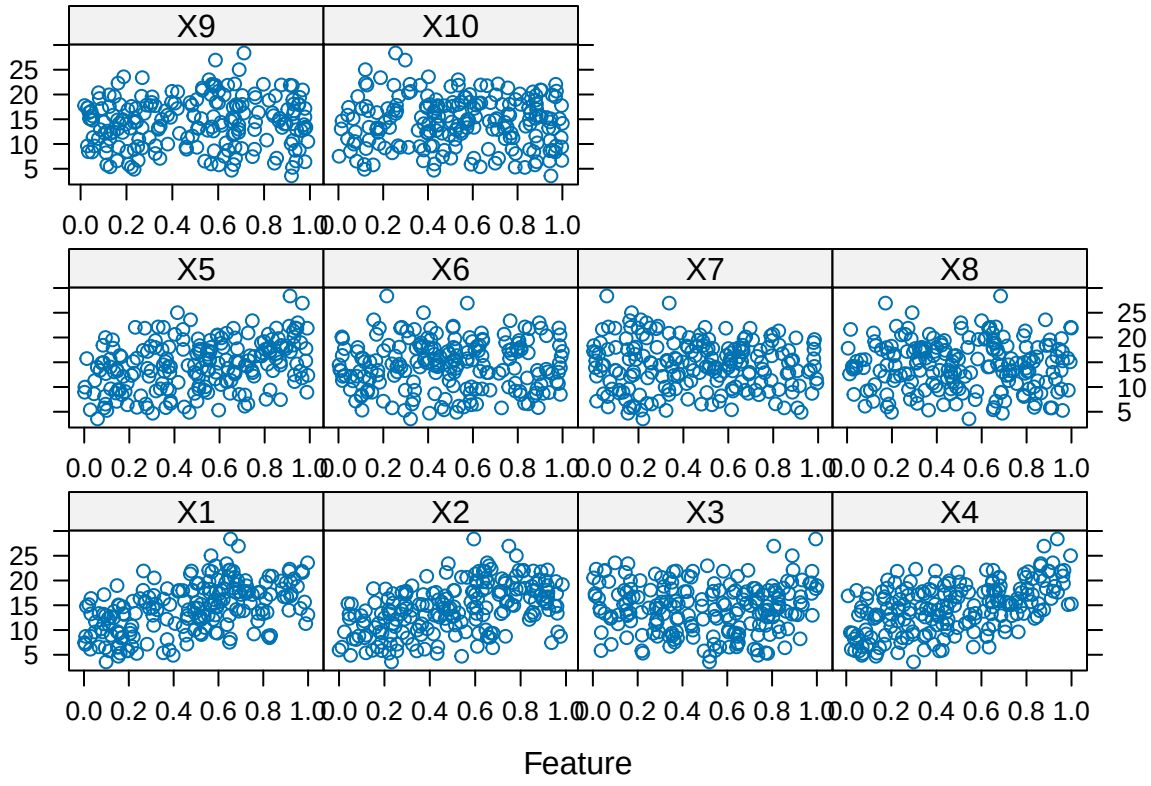
```
## corrplot 0.95 loaded
```

```
library(nnet)
```

```
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)
```

Tune several models on these data.

- We will train a few different non-linear models to the data.

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

- Answer: Mars model has the best performance with the highest R^2 of 0.8677298, and lowest RMSE 1.8136467. #### KNN Model

```
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
```

```
set.seed(1234)

nnetFit <- nnet(trainingData$x, trainingData$y,
                size = 5,
                decay = 0.01,
                linout = TRUE,
                trace = FALSE,
                maxit = 500,
                MaxNWts = 5 * (ncol(trainingData$x) + 1) +5 +1)

nnetPred <- predict(nnetFit, newdata = testData$x)
postResample(pred = nnetPred, obs = testData$y)
```

Neural Network

```
##      RMSE  Rsquared       MAE
## 2.5013620 0.7722019 1.9424460
```

```
marsFit <- earth(trainingData$x, trainingData$y)
marsFit
```

MARs

```
## 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
```

```
set.seed(1234)
marsPred <- predict(marsFit, newdata = testData$x)
postResample(pred = marsPred, obs = testData$y)
```

```
##      RMSE  Rsquared       MAE
## 1.8136467 0.8677298 1.3911836
```

```
svmRTuned <- train(trainingData$x, trainingData$y,
                   method = "svmRadial",
                   preProc = c("center","scale"),
                   tuneLength = 14,
                   trControl = trainControl(method = "cv"))

svmRTuned
```

SVMs

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   C        RMSE      Rsquared   MAE
##      0.25  2.488356  0.8012693  2.002699
##      0.50  2.236035  0.8154981  1.781068
##      1.00  2.069534  0.8361229  1.630495
##      2.00  1.965108  0.8519953  1.534286
##      4.00  1.881843  0.8628681  1.467129
##      8.00  1.829477  0.8699963  1.438937
##     16.00  1.815857  0.8723863  1.443647
##     32.00  1.816203  0.8723337  1.444164
##     64.00  1.816203  0.8723337  1.444164
##    128.00  1.816203  0.8723337  1.444164
##    256.00  1.816203  0.8723337  1.444164
##    512.00  1.816203  0.8723337  1.444164
##   1024.00  1.816203  0.8723337  1.444164
##   2048.00  1.816203  0.8723337  1.444164
##
## Tuning parameter 'sigma' was held constant at a value of 0.05870168
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.05870168 and C = 16.
```

```
set.seed(1234)
svmPred <- predict(svmRTuned, newdata = testData$x)
postResample(pred = svmPred, obs = testData$y)
```

```
##      RMSE  Rsquared       MAE
## 2.0614421 0.8276666 1.5664051
```

Exercise 7.5

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

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

- Answer: The SVM are the best performing model. SVM model has RMSE 0.6639696 and Rsquared 0.5548810.

```
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version 4.4.3
```

```r
data(ChemicalManufacturingProcess)
```

```r
sum(is.na(ChemicalManufacturingProcess))
```

```
## [1] 106
```

```r
## impute missing values
imputed_df <- preProcess(ChemicalManufacturingProcess, "knnImpute")
imp_df <- predict(imputed_df, ChemicalManufacturingProcess)
```

```r
sum(is.na(imp_df))
```

```
## [1] 0
```

```r
# split the data in 80/20
set.seed(123)
train_index <- createDataPartition(imp_df$Yield, p = 0.8, list=FALSE)

train_data <- imp_df[train_index, ]
test_data  <- imp_df[-train_index, ]

# separate predictors and response
X_train <- train_data[, -1]  # Remove Yield column
y_train <- train_data$Yield

X_test <- test_data[, -1]
y_test <- test_data$Yield
```

```r
set.seed(1234)
knnTune <- train(x = X_train,
                 y = y_train,
                 method = "knn",
                 preProcess = c('scale', 'center'),
                 tuneGrid = data.frame(.k = 1:20),
                 trControl = trainControl (method = "cv"))

knnTune
```

KNN Model

```
## k-Nearest Neighbors
##
## 144 samples
##  57 predictor
##
## Pre-processing: scaled (57), centered (57)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 128, 129, 129, 128, 131, 130, ...
```

```
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    1  0.7628737  0.4592019  0.5923371
##    2  0.6583938  0.5698937  0.5263008
##    3  0.6513992  0.5790112  0.5144975
##    4  0.6788856  0.5482130  0.5321965
##    5  0.6840773  0.5466160  0.5399008
##    6  0.6776610  0.5644737  0.5310965
##    7  0.6912231  0.5576849  0.5478956
##    8  0.7018618  0.5556077  0.5638154
##    9  0.7067132  0.5434831  0.5653327
##   10  0.7108070  0.5429782  0.5746481
##   11  0.7179731  0.5194462  0.5805881
##   12  0.7203870  0.5165982  0.5806824
##   13  0.7211139  0.5111691  0.5833027
##   14  0.7261908  0.5052909  0.5901406
##   15  0.7306338  0.4965346  0.5912589
##   16  0.7255281  0.5046651  0.5881941
##   17  0.7306463  0.5041565  0.5907774
##   18  0.7294443  0.5111512  0.5895441
##   19  0.7356724  0.5071330  0.5934638
##   20  0.7382388  0.5024346  0.5945759
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 3.
```

```r
# predict
knn_Tune_pred<- predict(knnTune, X_test)
postResample(knn_Tune_pred, y_test)
```

```
##      RMSE  Rsquared       MAE
## 0.7882800 0.3654029 0.6047951
```

```r
set.seed(1234)
nnetFit2 <- nnet(x = X_train,
                 y = y_train,
                 size = 5,
                 decay = 0.01,
                 linout = TRUE,
                 trace = FALSE,
                 maxit = 500,
                 MaxNWts = 5 * (ncol(X_train) + 1) +5 +1)

nnetPred2 <- predict(nnetFit2, X_test)
postResample(pred = nnetPred2, y_test)
```

Neural Network

```
##      RMSE  Rsquared       MAE
## 0.8320112 0.3847564 0.6393352
```

```r
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)
set.seed(1234)
mars_model2 <- train(x = X_train,
                     y = y_train,
                     method = "earth",
                     tuneGrid = marsGrid,
                     trControl = trainControl(method = "cv", number = 10))

mars_model2
```

MARs

```
## Multivariate Adaptive Regression Spline
##
## 144 samples
##  57 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 128, 129, 129, 128, 131, 130, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE       Rsquared   MAE
##   1        2      0.7675568  0.4738823  0.6121352
##   1        3      0.7516802  0.5447350  0.5705153
##   1        4      0.7497016  0.5553260  0.5628076
##   1        5      0.7779992  0.5421962  0.5703480
##   1        6      0.7693626  0.5328788  0.5762011
##   1        7      0.7474847  0.5525086  0.5674871
##   1        8      0.7651705  0.5427368  0.5792841
##   1        9      0.7831465  0.5257723  0.5897752
##   1       10      0.7936691  0.5221506  0.5905885
##   1       11      0.7951054  0.5235333  0.5864710
##   1       12      0.7840970  0.5399428  0.5775351
##   1       13      0.8046604  0.5093937  0.5940410
##   1       14      0.8066035  0.5155473  0.5877888
##   1       15      0.8119800  0.5079867  0.5933908
##   1       16      0.8153536  0.5008614  0.6005393
##   1       17      0.8166875  0.5001044  0.5982156
##   1       18      0.8168565  0.4951589  0.5931901
##   1       19      0.8177741  0.4957254  0.5971500
##   1       20      0.8158374  0.4969937  0.5941887
##   1       21      0.8169297  0.4974580  0.5923729
##   1       22      0.8163222  0.4982641  0.5903786
##   1       23      0.8182532  0.4989287  0.5907249
##   1       24      0.8177144  0.4992800  0.5904695
##   1       25      0.8177144  0.4992800  0.5904695
##   1       26      0.8177144  0.4992800  0.5904695
##   1       27      0.8177144  0.4992800  0.5904695
##   1       28      0.8177144  0.4992800  0.5904695
##   1       29      0.8177144  0.4992800  0.5904695
##   1       30      0.8177144  0.4992800  0.5904695
```

```
##   1       31      0.8177144  0.4992800  0.5904695
##   1       32      0.8177144  0.4992800  0.5904695
##   1       33      0.8177144  0.4992800  0.5904695
##   1       34      0.8177144  0.4992800  0.5904695
##   1       35      0.8177144  0.4992800  0.5904695
##   1       36      0.8177144  0.4992800  0.5904695
##   1       37      0.8177144  0.4992800  0.5904695
##   1       38      0.8177144  0.4992800  0.5904695
##   2        2      0.7675568  0.4738823  0.6121352
##   2        3      0.7029712  0.5197073  0.5676264
##   2        4      0.6423391  0.5980540  0.5146905
##   2        5      0.6642737  0.5633118  0.5200468
##   2        6      0.6709763  0.5610894  0.5177239
##   2        7      0.6887979  0.5601649  0.5209233
##   2        8      0.7125698  0.5394630  0.5275259
##   2        9      0.6873078  0.5613875  0.5106934
##   2       10      0.6810222  0.5761609  0.5057340
##   2       11      0.6829619  0.5726571  0.5077006
##   2       12      0.7125824  0.5442234  0.5277827
##   2       13      0.7117664  0.5617516  0.5294530
##   2       14      0.7235275  0.5620684  0.5241750
##   2       15      0.7143309  0.5489542  0.5296150
##   2       16      0.6914242  0.5672789  0.5137883
##   2       17      0.7233190  0.5421417  0.5366361
##   2       18      0.7456471  0.5337626  0.5422029
##   2       19      0.7283536  0.5428288  0.5336557
##   2       20      0.7871940  0.5370817  0.5637773
##   2       21      0.7581053  0.5410909  0.5531074
##   2       22      0.7690479  0.5420224  0.5563203
##   2       23      0.7678725  0.5426569  0.5537249
##   2       24      0.7678725  0.5426569  0.5537249
##   2       25      0.7678725  0.5426569  0.5537249
##   2       26      0.7678725  0.5426569  0.5537249
##   2       27      0.7678725  0.5426569  0.5537249
##   2       28      0.7678725  0.5426569  0.5537249
##   2       29      0.7678725  0.5426569  0.5537249
##   2       30      0.7678725  0.5426569  0.5537249
##   2       31      0.7678725  0.5426569  0.5537249
##   2       32      0.7678725  0.5426569  0.5537249
##   2       33      0.7678725  0.5426569  0.5537249
##   2       34      0.7678725  0.5426569  0.5537249
##   2       35      0.7678725  0.5426569  0.5537249
##   2       36      0.7678725  0.5426569  0.5537249
##   2       37      0.7678725  0.5426569  0.5537249
##   2       38      0.7678725  0.5426569  0.5537249
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 4 and degree = 2.
```

```
mars_pred2 <- predict(mars_model2, X_test)
postResample(mars_pred2, y_test)
```

```
##      RMSE  Rsquared       MAE
## 0.6842091 0.5900675 0.5622954
```

```
set.seed(123)
svmRTuned2 <- train(x = X_train,
                    y = y_train,
                    method = "svmRadial",
                    preProc = c("center","scale"),
                    tuneLength = 14,
                    trControl = trainControl(method = "cv"))

svmRTuned2
```

SVMs

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 144 samples
##  57 predictor
##
## Pre-processing: centered (57), scaled (57)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 128, 129, 129, 130, 128, 131, ...
## Resampling results across tuning parameters:
##
##   C        RMSE       Rsquared   MAE
##      0.25  0.7221020  0.5801363  0.5842048
##      0.50  0.6636429  0.6281290  0.5334680
##      1.00  0.6205969  0.6788291  0.4895894
##      2.00  0.6017919  0.6954916  0.4771769
##      4.00  0.6093532  0.6813266  0.4875209
##      8.00  0.5986302  0.6937148  0.4859324
##     16.00  0.5971711  0.6956875  0.4851146
##     32.00  0.5971711  0.6956875  0.4851146
##     64.00  0.5971711  0.6956875  0.4851146
##    128.00  0.5971711  0.6956875  0.4851146
##    256.00  0.5971711  0.6956875  0.4851146
##    512.00  0.5971711  0.6956875  0.4851146
##   1024.00  0.5971711  0.6956875  0.4851146
##   2048.00  0.5971711  0.6956875  0.4851146
##
## Tuning parameter 'sigma' was held constant at a value of 0.0146125
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.0146125 and C = 16.
```

```
set.seed(1234)
svmPred2 <- predict(svmRTuned2, X_test)
postResample(pred = svmPred2, y_test)
```

```
##      RMSE  Rsquared       MAE
## 0.6639696 0.5548810 0.5603961
```

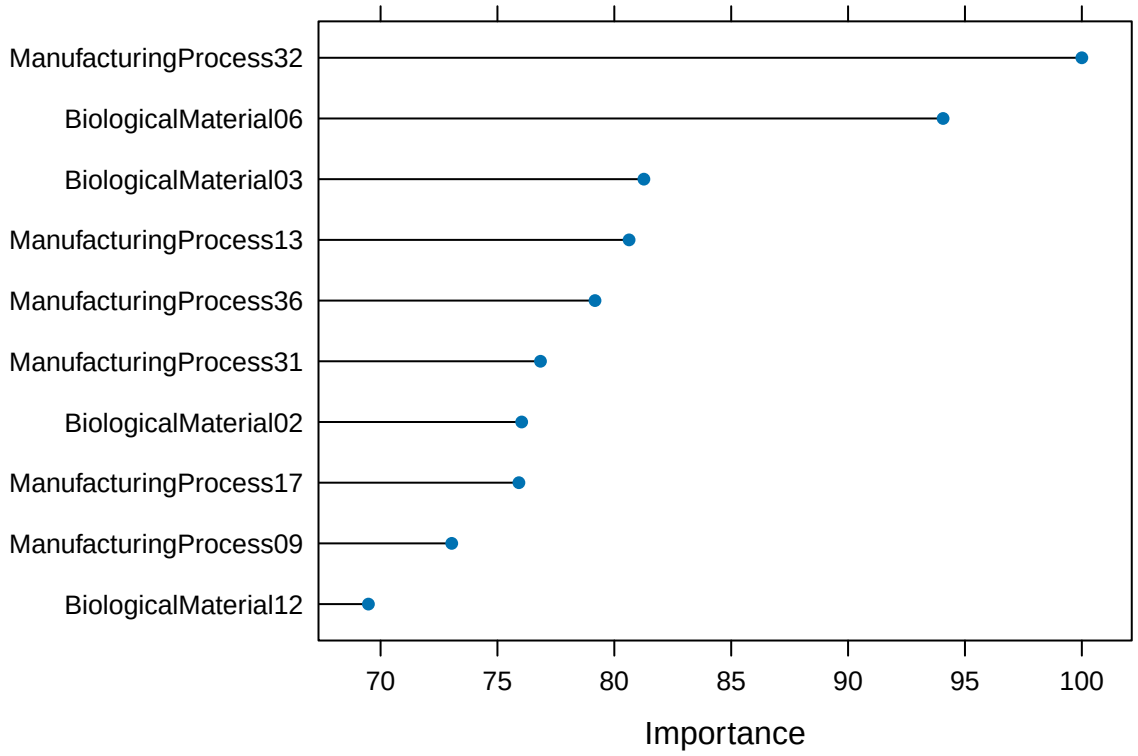
```
list(knn = postResample(knn_Tune_pred, y_test),
     nnet = postResample(pred = nnetPred2, y_test),
     MARS = postResample(mars_pred2, y_test),
     svm = postResample(pred = svmPred2, y_test))
```

```
## $knn
##      RMSE  Rsquared       MAE
## 0.7882800 0.3654029 0.6047951
##
## $nnet
##      RMSE  Rsquared       MAE
## 0.8320112 0.3847564 0.6393352
##
## $MARS
##      RMSE  Rsquared       MAE
## 0.6842091 0.5900675 0.5622954
##
## $svm
##      RMSE  Rsquared       MAE
## 0.6639696 0.5548810 0.5603961
```

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

- We can see in the plot, that ManufacturingProcess32, BiologicalMaterial06, BiologicalMaterial03, ManufacturingProcess13, ManufacturingProcess36, ManufacturingProcess31..ect are important to nonlinear SVM model.

```
plot(varImp(svmRTuned2), 10)
```



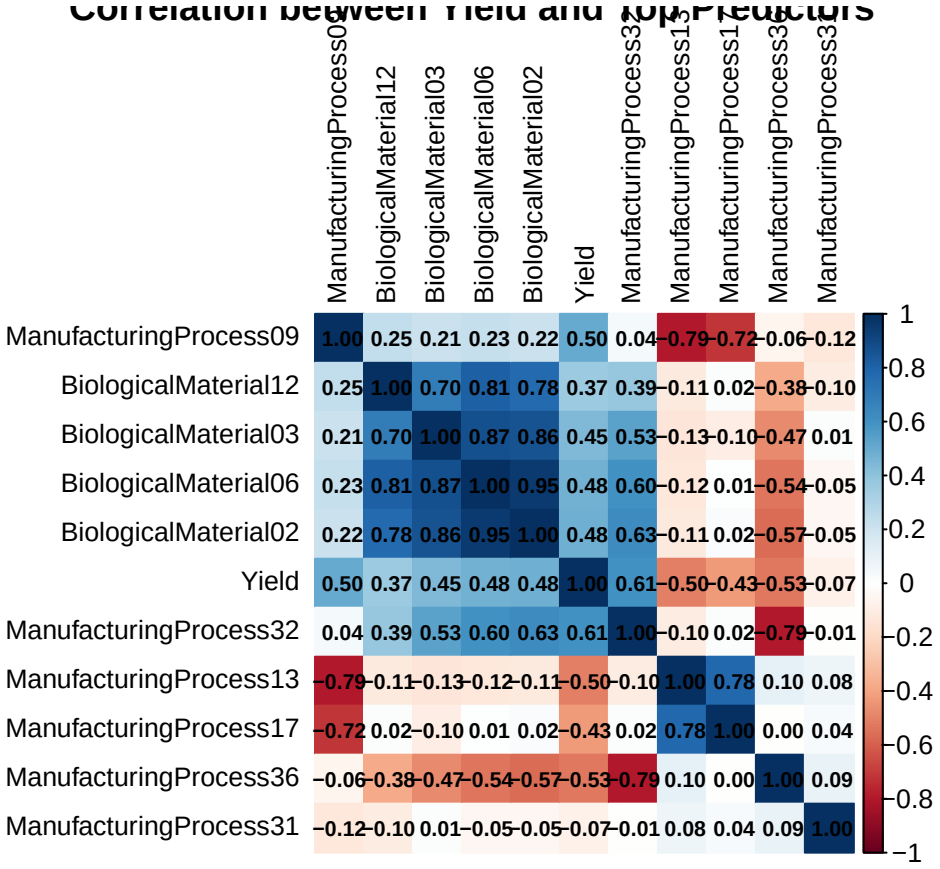
(c) Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

- Strongest positive correlations with Yield is ManufacturingProcess32 (0.61), followed by BiologicalMaterial03, and BiologicalMaterial02.

```
top_pre_names <- varImp(svmRTuned2)$importance %>%
  as.data.frame() %>%
  rownames_to_column('predictors') %>%
  slice_max(order_by = Overall, n = 10) %>%
  pull(predictors)
```

```
correlations <- imp_df %>%
  select(all_of(c("Yield", top_pre_names))) %>%
  cor()

corrplot(correlations,
         method = "color",
         order = "hclust",
         addCoef.col = "black",
         tl.col = "black",
         tl.cex = 0.8,
         number.cex = 0.7,
         main = "Correlation between Yield and Top Predictors")
```


Correlation between Yield and Top Predictors
ManufacturingProcess09
BiologicalMaterial12
BiologicalMaterial03
BiologicalMaterial06
BiologicalMaterial02
Yield
ManufacturingProcess32
ManufacturingProcess13
ManufacturingProcess17
ManufacturingProcess36
ManufacturingProcess31
1.00 0.25 0.21 0.23 0.22 0.50 0.04 −0.79 −0.72 −0.06 −0.12
0.25 1.00 0.70 0.81 0.78 0.37 0.39 −0.11 0.02 −0.38 −0.10
0.21 0.70 1.00 0.87 0.86 0.45 0.53 −0.13 −0.10 −0.47 0.01
0.23 0.81 0.87 1.00 0.95 0.48 0.60 −0.12 0.01 −0.54 −0.05
0.22 0.78 0.86 0.95 1.00 0.48 0.63 −0.11 0.02 −0.57 −0.05
0.50 0.37 0.45 0.48 0.48 1.00 0.61 −0.50 −0.43 −0.53 −0.07
0.04 0.39 0.53 0.60 0.63 0.61 1.00 −0.10 0.02 −0.79 −0.01
−0.79 −0.11 −0.13 −0.12 −0.11 −0.50 −0.10 1.00 0.78 0.10 0.08
−0.72 0.02 −0.10 0.01 0.02 −0.43 0.02 0.78 1.00 0.00 0.04
−0.06 −0.38 −0.47 −0.54 −0.57 −0.53 −0.79 0.10 0.00 1.00 0.09
−0.12 −0.10 0.01 −0.05 −0.05 −0.07 −0.01 0.08 0.04 0.09 1.00
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1