Team 2 - Homework Two

Assignment 2: KJ 7.2; KJ 7.5

Sang Yoon (Andy) Hwang

DATE:2019-11-01

Dependencies

```
# predictive modeling
libraries('mlbench', 'caret', 'AppliedPredictiveModeling')

# Formatting Libraries
libraries('default', 'knitr', 'kableExtra')

# Plotting Libraries
libraries('ggplot2', 'grid', 'ggfortify')
```

(1) Kuhn & Johnson 7.2

> Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data: $y = 10\sin(\pi x_1 x_2)20(x_3 - 0.5)^2 10x_4 5x_5 N(0, \sigma^2)$; where the x values are random variables uniformly distributed between $[0, 1]$ (there are also 5 other non-informative variables also created in the simulation).

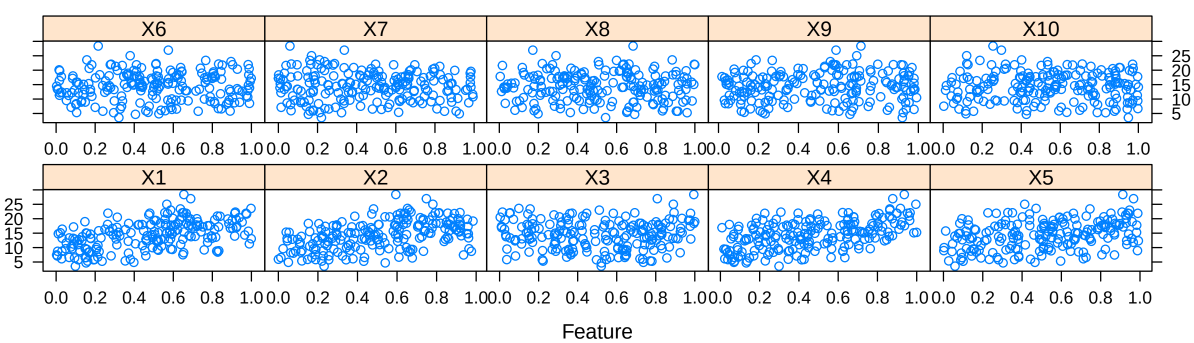
The package `mlbench` contains a function called `mlbench.friedman1` that simulates these data:

```
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)
```

(a) Tune several models on these data. For example:

```
set.seed(200)
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  3.554554  0.4895311  2.883670
   7  3.423303  0.5264402  2.767070
   9  3.361439  0.5525056  2.702852
  11  3.275234  0.5885952  2.632363
  13  3.245376  0.6099949  2.607849
  15  3.218637  0.6308597  2.576730
  17  3.229692  0.6380326  2.589076
  19  3.231915  0.6463092  2.595749
  21  3.228217  0.6591640  2.599618
  23  3.254794  0.6610119  2.628392

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 15.
```

```
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
3.1750657 0.6785946 2.5443169
```

Model 1: KNN model with hyperparameter tuning

```
set.seed(100)
knn_model <- train(trainingData$x,
 trainingData$y,
 method = "knn",
 # Center and scaling will occur for new predictions too
```

```r
preProc = c("center", "scale"),
tuneGrid = data.frame(.k = 1:50),
trControl = trainControl(method = "cv"))

knn_model
```

```
k-Nearest Neighbors

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:

  k   RMSE      Rsquared   MAE
   1  4.218081  0.4124441  3.500924
   2  3.546791  0.5197988  2.941632
   3  3.460792  0.5170308  2.841547
   4  3.304737  0.5584950  2.697459
   5  3.281638  0.5736341  2.695889
   6  3.201700  0.6056758  2.651120
   7  3.212544  0.6083203  2.677007
   8  3.175321  0.6293337  2.589611
   9  3.117499  0.6632514  2.555966
  10  3.118730  0.6619864  2.542520
  11  3.059105  0.6860779  2.486276
  12  3.110398  0.6794240  2.539603
  13  3.107626  0.6909955  2.504201
  14  3.098502  0.6981615  2.503473
  15  3.107763  0.7020139  2.517724
  16  3.092651  0.7141572  2.499280
  17  3.094870  0.7184612  2.502765
  18  3.106069  0.7205098  2.512843
  19  3.110156  0.7223952  2.522663
  20  3.129738  0.7248629  2.544792
  21  3.125325  0.7282228  2.555344
  22  3.152564  0.7265154  2.583124
  23  3.156426  0.7299184  2.590130
  24  3.156312  0.7315798  2.583509
  25  3.168639  0.7320840  2.600023
  26  3.192311  0.7301789  2.615933
  27  3.191953  0.7359772  2.626247
  28  3.236432  0.7229397  2.667947
  29  3.266908  0.7199837  2.696586
  30  3.264200  0.7279598  2.695916
  31  3.273003  0.7273885  2.702978
  32  3.282489  0.7276198  2.702930
  33  3.289932  0.7284479  2.707697
  34  3.303444  0.7281557  2.717839
  35  3.320845  0.7249744  2.727184
  36  3.331038  0.7257760  2.729478
  37  3.342847  0.7296750  2.740671
```

```
  38  3.353494  0.7305494  2.745450
  39  3.363513  0.7335993  2.748193
  40  3.369217  0.7349916  2.752439
  41  3.394484  0.7291182  2.779272
  42  3.404124  0.7292558  2.783362
  43  3.403433  0.7342789  2.780712
  44  3.418392  0.7341581  2.794006
  45  3.420892  0.7410895  2.790850
  46  3.421315  0.7485185  2.801475
  47  3.434838  0.7498270  2.816174
  48  3.449996  0.7487060  2.831474
  49  3.451778  0.7525105  2.833605
  50  3.467105  0.7471964  2.844317

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 11.
```

```
knn_Pred <- predict(knn_model, newdata = testData$x)

## The function 'postResample' can be used to get the test set performance values
knn_pv <- postResample(pred = knn_Pred, obs = testData$y)
knn_pv
```

```
     RMSE  Rsquared       MAE
3.1222641 0.6690472 2.4963650
```

Unlike above approach where `tuneLength` = 10 to find 10 odd numbered Ks starting from 5, we will set `tuneGrid` running from k = 1 to 50 after CV process. RMSE on validation set was used to select the optimal model using the smallest value. The final value used for the model was k = 11 with RMSE on test set of 3.1222641.

Model 2: Neural Networks

```
# remove highly correlated predictors to ensure that the maximum absolute pariwise correlation between
# we did not have any highly correlated predictors so let's keep the features as they are.
findCorrelation(cor(trainingData$x), cutoff = .75)
```

```
integer(0)
```

```
# hyperparameter tuning for nnet
nnetGrid <- expand.grid(.size = c(1:10), .decay = c(0, 0.01, .1))

set.seed(100)
nnet_model <- train(trainingData$x, trainingData$y,
 method = "nnet",
 tuneGrid = nnetGrid,
 trControl = trainControl(method="cv"),
 ## Automatically standardize data prior to modeling and prediction
 preProc = c("center", "scale"),
 linout = TRUE,
 trace = FALSE,
 MaxNWts = 10 * (ncol(trainingData$x) + 1)  + 10 +  1,
 maxit = 500)

nnet_model
```

```
Neural Network
```

```
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:

  size  decay  RMSE      Rsquared   MAE
   1    0.00   2.803056  0.6666327  2.258691
   1    0.01   2.427596  0.7621743  1.887808
   1    0.10   2.435471  0.7608686  1.890066
   2    0.00   2.489139  0.7430098  1.959395
   2    0.01   2.580965  0.7303017  2.026386
   2    0.10   2.697160  0.7081857  2.145994
   3    0.00   2.175481  0.8114073  1.736758
   3    0.01   2.247174  0.7923587  1.847530
   3    0.10   2.574029  0.7395256  2.056691
   4    0.00   2.338196  0.7863493  1.857565
   4    0.01   2.382258  0.7789730  1.875869
   4    0.10   2.441937  0.7640421  1.906016
   5    0.00   4.095639  0.6461716  2.815675
   5    0.01   2.611351  0.7275315  2.058338
   5    0.10   2.527299  0.7479751  2.043740
   6    0.00   4.360771  0.5655777  2.776134
   6    0.01   2.740151  0.7334422  2.120769
   6    0.10   2.723467  0.7037555  2.191856
   7    0.00   8.107456  0.6013541  3.614263
   7    0.01   2.634878  0.7306846  2.082135
   7    0.10   2.634554  0.7272090  2.034430
   8    0.00   9.294033  0.4764955  3.995120
   8    0.01   3.239373  0.6734412  2.543492
   8    0.10   2.857949  0.7043441  2.285976
   9    0.00   7.318398  0.5467571  3.670600
   9    0.01   3.541729  0.5379448  2.671080
   9    0.10   3.187206  0.6050659  2.544700
  10    0.00   4.083062  0.5181907  2.838082
  10    0.01   3.525950  0.5879990  2.836265
  10    0.10   3.010387  0.6719106  2.387353

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 3 and decay = 0.
```

```r
nnet_Pred <- predict(nnet_model, newdata = testData$x)

## The function 'postResample' can be used to get the test set performance values
nnet_pv <- postResample(pred = nnet_Pred, obs = testData$y)
nnet_pv
```

```
     RMSE  Rsquared       MAE
2.3950120 0.7740742 1.7970761
```

We found nnet with `size` = 3 (number of units in the hidden layer) and `decay` = 0 (parameter for weight decay) is the optimal model based on RMSE on validating set. RMSE on test set was 2.395012.

Model 3: Neural Networks Using Model Averaging

```r
# hyperparameter tuning for avnnet
nnetGrid2 <- expand.grid(.size = c(1:10), .decay = c(0, 0.01, .1), .bag = FALSE)

set.seed(100)
avnnet_model <- train(trainingData$x, trainingData$y,
 method = "avNNet",
 tuneGrid = nnetGrid2,
 trControl = trainControl(method="cv"),
 ## Automatically standardize data prior to modeling and prediction
 preProc = c("center", "scale"),
 linout = TRUE,
 trace = FALSE,
 MaxNWts = 10 * (ncol(trainingData$x) + 1)  + 10 +  1,
 maxit = 500)

avnnet_model
```

```
Model Averaged Neural Network

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:

  size  decay  RMSE      Rsquared   MAE
   1    0.00   2.451722  0.7625351  1.896021
   1    0.01   2.427658  0.7621763  1.887707
   1    0.10   2.435387  0.7608838  1.890037
   2    0.00   2.458303  0.7556215  1.947825
   2    0.01   2.447743  0.7533484  1.884509
   2    0.10   2.483261  0.7506467  1.924237
   3    0.00   2.086103  0.8297667  1.625836
   3    0.01   2.155718  0.8096710  1.676070
   3    0.10   2.227768  0.8003473  1.734663
   4    0.00   2.036944  0.8294847  1.636994
   4    0.01   2.102926  0.8232439  1.623997
   4    0.10   2.032960  0.8301759  1.604834
   5    0.00   2.314361  0.7871277  1.717315
   5    0.01   2.131569  0.8131597  1.670786
   5    0.10   2.192707  0.8054839  1.778156
   6    0.00   2.593804  0.7448382  1.966474
   6    0.01   2.167612  0.8108275  1.674682
   6    0.10   2.042238  0.8227937  1.645410
   7    0.00   4.989121  0.5232150  3.110796
   7    0.01   2.238776  0.7948069  1.754542
   7    0.10   2.202599  0.7926956  1.751645
   8    0.00   5.839958  0.5788055  3.266803
   8    0.01   2.392621  0.7727856  1.893142
   8    0.10   2.271702  0.7860966  1.790783
   9    0.00   5.138515  0.4191673  3.248740
```

```
   9    0.01   2.463813  0.7534985  1.927194
   9    0.10   2.250755  0.7930757  1.820346
  10    0.00   3.400511  0.6426513  2.428742
  10    0.01   2.456849  0.7404324  1.988814
  10    0.10   2.437919  0.7608963  1.971687

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 = 4, decay = 0.1 and bag
 = FALSE.
```

```
avnnet_Pred <- predict(avnnet_model, newdata = testData$x)

## The function 'postResample' can be used to get the test set performance values
avnnet_pv <- postResample(pred = avnnet_Pred, obs = testData$y)
avnnet_pv
```

```
     RMSE  Rsquared       MAE 
2.1306481 0.8202697 1.5982639 
```

We found nnet with `size` = 4 (number of units in the hidden layer) and `decay` = 0.1 (parameter for weight decay) is the optimal model based on RMSE on validating set. RMSE on test set was 2.1306481.

Model 4: Multivariate Adaptive Regression Splines

```
# hyperparameter tuning for MARS
marsGrid <- expand.grid(.degree = 1:3, .nprune = 2:38)

set.seed(100)
mars_model <- train(trainingData$x, trainingData$y,
 method = "earth",
 tuneGrid = marsGrid,
 trControl = trainControl(method="cv"))

#mars_model
summary(mars_model)
```

```
Call: earth(x=data.frame[200,10], y=c(18.46,16.1,17...), keepxy=TRUE,
            degree=2, nprune=17)

                                coefficients
(Intercept)                        20.378441
h(0.621722-X1)                    -15.512132
h(X1-0.621722)                      9.177132
h(0.601063-X2)                    -17.940676
h(X2-0.601063)                     10.064356
h(X3-0.281766)                     11.590022
h(0.447442-X3)                     14.641640
h(X3-0.447442)                    -12.924806
h(X3-0.606015)                     13.416764
h(0.734892-X4)                    -10.074386
h(X4-0.734892)                      9.687149
h(0.850094-X5)                     -5.385762
h(0.218266-X1) * h(X2-0.601063)   -55.372637
h(X1-0.218266) * h(X2-0.601063)   -27.542831
h(X1-0.621722) * h(X2-0.295997)   -26.527403
```

```
h(0.649253-X1) * h(0.601063-X2)    26.129827

Selected 16 of 18 terms, and 5 of 10 predictors
Termination condition: Reached nk 21
Importance: X1, X4, X2, X5, X3, X6-unused, X7-unused, X8-unused, ...
Number of terms at each degree of interaction: 1 11 4
GCV 1.61518    RSS 210.6377    GRSq 0.934423    RSq 0.9568093
```

```
mars_Pred <- predict(mars_model, newdata = testData$x)

## The function 'postResample' can be used to get the test set performance values
mars_pv <- postResample(pred = mars_Pred, obs = testData$y)
mars_pv
```

```
     RMSE  Rsquared       MAE
1.1492504 0.9471145 0.9158382
```

We found MARS with `degree` = 2 (Maximum degree of interaction (Friedman's mi)) and `nprune` = 17 (aximum number of terms (including intercept) in the pruned model) is the optimal model based on RMSE on validating set. RMSE on test set was 1.1492504.

Model 5: Support Vector regression

```
set.seed(100)
svm_model <- train(trainingData$x, trainingData$y,
 method = "svmRadial",
 preProc = c("center", "scale"),
 tuneLength = 14,
 trControl = trainControl(method="cv"))

#svm_model
svm_model
```

```
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  2.534788  0.7882081  2.034824
    0.50  2.292127  0.8029516  1.819981
    1.00  2.091598  0.8284381  1.657402
    2.00  1.967193  0.8457471  1.546737
    4.00  1.883133  0.8561761  1.482054
    8.00  1.863807  0.8588797  1.468328
   16.00  1.834215  0.8633819  1.456738
   32.00  1.836471  0.8632508  1.459909
   64.00  1.836471  0.8632508  1.459909
  128.00  1.836471  0.8632508  1.459909
  256.00  1.836471  0.8632508  1.459909
  512.00  1.836471  0.8632508  1.459909
```

```
  1024.00  1.836471  0.8632508  1.459909
  2048.00  1.836471  0.8632508  1.459909

Tuning parameter 'sigma' was held constant at a value of 0.0552698
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.0552698 and C = 16.
```

```r
svm_Pred <- predict(svm_model, newdata = testData$x)

## The function 'postResample' can be used to get the test set performance values
svm_pv <- postResample(pred = svm_Pred, obs = testData$y)
svm_pv
```

```
     RMSE  Rsquared       MAE
2.0490047 0.8297577 1.5586106
```

Since the nature of the equation of the data is non-linear, we will use `svmRadial` as kernal function for regression. The final values used for the model were sigma = 0.0552698 and C = 16 with RMSE on test set of 2.0490047.

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

MARS appears to give the best performance based on RMSE, R squared and MAE on test set. The summary out put of `mars_model` gives us that `Importance: X1, X4, X2, X5, X3, X6-unused, X7-unused, X8-unused, X9-unused, ...`. MARS does select the informative predictors X1-X5 only.

```r
#code
# Model performance metrics
sum_t <- data.frame(
            knn_pv,
            nnet_pv,
            avnnet_pv,
            mars_pv,
            svm_pv
            )
print(sum_t)
```

```
            knn_pv   nnet_pv avnnet_pv   mars_pv    svm_pv
RMSE     3.1222641 2.3950120 2.1306481 1.1492504 2.0490047
Rsquared 0.6690472 0.7740742 0.8202697 0.9471145 0.8297577
MAE      2.4963650 1.7970761 1.5982639 0.9158382 1.5586106
```

```r
# summary mars
summary(mars_model)
```

```
Call: earth(x=data.frame[200,10], y=c(18.46,16.1,17...), keepxy=TRUE,
            degree=2, nprune=17)

                               coefficients
(Intercept)                       20.378441
h(0.621722-X1)                   -15.512132
h(X1-0.621722)                     9.177132
h(0.601063-X2)                   -17.940676
h(X2-0.601063)                    10.064356
h(X3-0.281766)                    11.590022
h(0.447442-X3)                    14.641640
h(X3-0.447442)                   -12.924806
```

```
h(X3-0.606015)                    13.416764
h(0.734892-X4)                   -10.074386
h(X4-0.734892)                     9.687149
h(0.850094-X5)                    -5.385762
h(0.218266-X1) * h(X2-0.601063)   -55.372637
h(X1-0.218266) * h(X2-0.601063)   -27.542831
h(X1-0.621722) * h(X2-0.295997)   -26.527403
h(0.649253-X1) * h(0.601063-X2)    26.129827

Selected 16 of 18 terms, and 5 of 10 predictors
Termination condition: Reached nk 21
Importance: X1, X4, X2, X5, X3, X6-unused, X7-unused, X8-unused, ...
Number of terms at each degree of interaction: 1 11 4
GCV 1.61518    RSS 210.6377    GRSq 0.934423    RSq 0.9568093
```

(2) Kuhn & Johnson 7.5

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

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

```r
# code
# split data train/test
training <- df_final$Yield %>%
  createDataPartition(p = 0.8, list = FALSE)

df_train  <- df_final[training, ]
df_test <- df_final[-training, ]

# model1 - KNN
set.seed(100)
knn_model2 <- train(Yield~., data = df_train,
                   method = "knn",
                   # Center and scaling will occur for new predictions too
                   preProc = c("center", "scale"),
                   tuneGrid = data.frame(.k = 1:50),
                   trControl = trainControl(method = "cv"))

knn_model2
```

```
k-Nearest Neighbors

144 samples
 56 predictor

Pre-processing: centered (56), scaled (56)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...
Resampling results across tuning parameters:

  k   RMSE      Rsquared   MAE
   1  1.562947  0.4104086  1.211757
   2  1.394114  0.4526620  1.103294
```

```
   3  1.285798  0.5202328  1.022250
   4  1.261951  0.5511694  1.005031
   5  1.290154  0.5346045  1.016046
   6  1.285469  0.5467205  1.015962
   7  1.277158  0.5691222  1.014170
   8  1.280950  0.5672804  1.022692
   9  1.260191  0.5951063  1.008777
  10  1.268148  0.5927028  1.022267
  11  1.274834  0.5762849  1.040308
  12  1.283879  0.5691015  1.046630
  13  1.300146  0.5565749  1.063440
  14  1.294960  0.5643696  1.054302
  15  1.315419  0.5533099  1.071270
  16  1.322033  0.5432361  1.078412
  17  1.333203  0.5374531  1.078099
  18  1.345231  0.5267062  1.089399
  19  1.354941  0.5240723  1.095199
  20  1.361053  0.5260280  1.100479
  21  1.357930  0.5291765  1.095387
  22  1.360462  0.5363972  1.100225
  23  1.369994  0.5307997  1.105951
  24  1.381541  0.5262263  1.116182
  25  1.390061  0.5225105  1.124752
  26  1.390820  0.5240722  1.125271
  27  1.401032  0.5200972  1.133713
  28  1.406645  0.5247034  1.137767
  29  1.417484  0.5124727  1.145324
  30  1.422851  0.5160566  1.149150
  31  1.424039  0.5158770  1.155165
  32  1.431312  0.5132871  1.164555
  33  1.441439  0.5029494  1.170119
  34  1.445577  0.5033464  1.173391
  35  1.446068  0.4997379  1.173717
  36  1.451050  0.4987082  1.175739
  37  1.450722  0.5073898  1.176310
  38  1.452006  0.5093112  1.180577
  39  1.458957  0.5080230  1.185948
  40  1.462371  0.5046533  1.190404
  41  1.470614  0.5009788  1.197493
  42  1.476481  0.4945194  1.201055
  43  1.478755  0.4983923  1.199669
  44  1.484498  0.4951622  1.203632
  45  1.490334  0.4982518  1.208023
  46  1.491714  0.5057906  1.210398
  47  1.496813  0.5022176  1.215699
  48  1.500448  0.5047809  1.219058
  49  1.506809  0.5048347  1.225709
  50  1.509769  0.5034198  1.228911

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 9.
```

```r
knn_Pred2 <- predict(knn_model2, newdata = df_test)
```

```
## The function 'postResample' can be used to get the test set performance values
knn_pv2 <- postResample(pred = knn_Pred2, obs = df_test$Yield)


# model2 - nnet
# remove highly correlated predictors to ensure that the maximum absolute pariwise correlation between
df_train_x <- df_train[-1]
df_train_y <- df_train[,1]
df_test_x <- df_test[-1]
df_test_y <- df_test[,1]

tooHigh <- findCorrelation(cor(df_train_x), cutoff = .75)

trainx_nn <- df_train_x[, -tooHigh]
testx_nn <-  df_test_x[, -tooHigh]

# hyperparameter tuning for nnet
nnetGrid12 <- expand.grid(.size = c(1:10), .decay = c(0, 0.01, .1))

set.seed(100)
nnet_model2 <- train(trainx_nn, df_train_y,
                    method = "nnet",
                    tuneGrid = nnetGrid12,
                    trControl = trainControl(method="cv"),
                    ## Automatically standardize data prior to modeling and prediction
                    preProc = c("center", "scale"),
                    linout = TRUE,
                    trace = FALSE,
                    MaxNWts = 10 * (ncol(trainx_nn) + 1)  + 10 +  1,
                    maxit = 500)

nnet_model2
```

```
Neural Network

144 samples
 35 predictor

Pre-processing: centered (35), scaled (35)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...
Resampling results across tuning parameters:

  size  decay  RMSE      Rsquared   MAE
   1    0.00   1.755205  0.2066616  1.455577
   1    0.01   1.531002  0.4448691  1.192716
   1    0.10   1.373454  0.5175944  1.109371
   2    0.00   2.212948  0.1773784  1.727733
   2    0.01   1.864348  0.3910941  1.542705
   2    0.10   1.715287  0.4696443  1.333888
   3    0.00   4.213947  0.2706156  3.025339
   3    0.01   3.402447  0.1312987  2.650608
   3    0.10   2.274638  0.3535430  1.809244
   4    0.00   3.753927  0.1694120  2.927575
```

```
   4    0.01    3.032265  0.2730534  2.293653
   4    0.10    2.662301  0.2596654  2.073559
   5    0.00    3.193235  0.2142524  2.523091
   5    0.01    2.847820  0.2693243  2.167825
   5    0.10    2.843172  0.2010863  2.136950
   6    0.00    3.141860  0.3508280  2.459695
   6    0.01    2.535074  0.1739787  2.089114
   6    0.10    2.009758  0.3223722  1.549843
   7    0.00    5.113950  0.1935983  3.377113
   7    0.01    2.849091  0.2545805  2.129850
   7    0.10    2.114057  0.2762636  1.594852
   8    0.00    4.294256  0.1767868  3.299536
   8    0.01    2.795623  0.1388297  2.169489
   8    0.10    2.275122  0.2983466  1.713263
   9    0.00    9.006573  0.2327137  5.799823
   9    0.01    2.730389  0.2517318  2.214361
   9    0.10    2.237043  0.3159019  1.714727
  10    0.00    8.689726  0.3538469  5.208147
  10    0.01    4.329962  0.1650993  3.119468
  10    0.10    2.410800  0.2083370  1.808763

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 1 and decay = 0.1.
```

```
nnet_Pred2 <- predict(nnet_model2, newdata = testx_nn)

## The function 'postResample' can be used to get the test set performance values
nnet_pv2 <-postResample(pred = nnet_Pred2, obs = df_test_y)

# model 3 - avNNet
# hyperparameter tuning for avnnet
nnetGrid22 <- expand.grid(.size = c(1:10), .decay = c(0, 0.01, .1), .bag = FALSE)

set.seed(100)
avnnet_model2 <- train(trainx_nn, df_train_y,
                      method = "avNNet",
                      tuneGrid = nnetGrid22,
                      trControl = trainControl(method="cv"),
                      ## Automatically standardize data prior to modeling and prediction
                      preProc = c("center", "scale"),
                      linout = TRUE,
                      trace = FALSE,
                      MaxNWts = 10 * (ncol(trainx_nn) + 1)  + 10 +  1,
                      maxit = 500)

avnnet_model2
```

```
Model Averaged Neural Network

144 samples
 35 predictor

Pre-processing: centered (35), scaled (35)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...
```

```
Resampling results across tuning parameters:

  size  decay  RMSE      Rsquared   MAE
   1    0.00   1.589451  0.3851410  1.271954
   1    0.01   1.376260  0.5016400  1.103760
   1    0.10   1.298216  0.5536547  1.053173
   2    0.00   1.614710  0.4230801  1.274696
   2    0.01   1.435679  0.5191818  1.116245
   2    0.10   1.486560  0.4888622  1.185787
   3    0.00   1.893573  0.3390867  1.482814
   3    0.01   1.947427  0.2967394  1.543828
   3    0.10   1.702341  0.4794429  1.298251
   4    0.00   2.183278  0.2198895  1.802826
   4    0.01   1.837276  0.3676411  1.438189
   4    0.10   1.841018  0.3934255  1.451878
   5    0.00   1.788868  0.3797231  1.487272
   5    0.01   1.677605  0.4073755  1.350633
   5    0.10   1.790401  0.4160895  1.340008
   6    0.00   2.183750  0.3741136  1.680205
   6    0.01   1.474744  0.5088465  1.228068
   6    0.10   1.842838  0.4122807  1.340819
   7    0.00   2.535701  0.3775290  2.006750
   7    0.01   1.678518  0.4521095  1.327766
   7    0.10   1.808841  0.3698767  1.370640
   8    0.00   3.327318  0.2290038  2.309492
   8    0.01   1.619979  0.4401995  1.268287
   8    0.10   1.789302  0.4025716  1.399899
   9    0.00   4.247876  0.2434059  2.861125
   9    0.01   1.699004  0.4937072  1.303050
   9    0.10   1.967303  0.2727353  1.492640
  10    0.00   7.011010  0.2117344  4.342906
  10    0.01   2.038958  0.3644072  1.468109
  10    0.10   1.748484  0.3668079  1.353886

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 = 1, decay = 0.1 and bag
 = FALSE.
```

```r
avnnet_Pred2 <- predict(avnnet_model2, newdata = testx_nn)

## The function 'postResample' can be used to get the test set performance values
avnnet_pv2 <- postResample(pred = avnnet_Pred2, obs = df_test_y)

# model 4 - MARS
# hyperparameter tuning for MARS
marsGrid2 <- expand.grid(.degree = 1:3, .nprune = 2:38)

set.seed(100)
mars_model2 <- train(Yield~., data = df_train,
                     method = "earth",
                     tuneGrid = marsGrid2,
                     trControl = trainControl(method="cv"))
```

```r
#mars_model
mars_model2
```

```
Multivariate Adaptive Regression Spline

144 samples
 56 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...
Resampling results across tuning parameters:

  degree  nprune  RMSE      Rsquared   MAE
  1        2      1.407400  0.4656139  1.1174779
  1        3      1.223427  0.6194501  0.9868635
  1        4      1.199954  0.6230092  0.9636997
  1        5      1.164765  0.6453235  0.9463908
  1        6      1.150927  0.6495515  0.9307132
  1        7      1.098522  0.6828422  0.8871423
  1        8      1.150254  0.6602017  0.9190847
  1        9      1.163031  0.6570934  0.9352294
  1       10      1.202534  0.6312059  0.9743665
  1       11      1.185338  0.6391984  0.9467194
  1       12      1.176559  0.6346864  0.9469068
  1       13      1.211428  0.6172954  0.9842769
  1       14      1.217020  0.6188990  0.9886593
  1       15      1.223154  0.6160753  0.9926387
  1       16      1.223154  0.6160753  0.9926387
  1       17      1.223154  0.6160753  0.9926387
  1       18      1.223154  0.6160753  0.9926387
  1       19      1.223154  0.6160753  0.9926387
  1       20      1.223154  0.6160753  0.9926387
  1       21      1.223154  0.6160753  0.9926387
  1       22      1.223154  0.6160753  0.9926387
  1       23      1.223154  0.6160753  0.9926387
  1       24      1.223154  0.6160753  0.9926387
  1       25      1.223154  0.6160753  0.9926387
  1       26      1.223154  0.6160753  0.9926387
  1       27      1.223154  0.6160753  0.9926387
  1       28      1.223154  0.6160753  0.9926387
  1       29      1.223154  0.6160753  0.9926387
  1       30      1.223154  0.6160753  0.9926387
  1       31      1.223154  0.6160753  0.9926387
  1       32      1.223154  0.6160753  0.9926387
  1       33      1.223154  0.6160753  0.9926387
  1       34      1.223154  0.6160753  0.9926387
  1       35      1.223154  0.6160753  0.9926387
  1       36      1.223154  0.6160753  0.9926387
  1       37      1.223154  0.6160753  0.9926387
  1       38      1.223154  0.6160753  0.9926387
  2        2      1.407400  0.4656139  1.1174779
  2        3      1.280199  0.5709792  1.0181681
  2        4      1.251083  0.5829880  1.0166939
```

2	5	1.194031	0.6200359	0.9658520
2	6	1.179059	0.6219074	0.9538453
2	7	1.091866	0.6786885	0.8650013
2	8	1.144683	0.6540160	0.9054151
2	9	1.199698	0.6234861	0.9292222
2	10	1.207126	0.6211522	0.9260997
2	11	1.278838	0.6183028	0.9523748
2	12	1.304635	0.6120710	0.9668832
2	13	1.339764	0.6048324	0.9990698
2	14	1.410092	0.5885196	1.0481210
2	15	1.411828	0.5860411	1.0611942
2	16	1.750442	0.5340995	1.1622914
2	17	1.786237	0.5342503	1.1606896
2	18	1.781165	0.5339347	1.1559614
2	19	1.783391	0.5384735	1.1460005
2	20	1.796314	0.5288854	1.1652398
2	21	2.062368	0.5207735	1.2547404
2	22	2.088379	0.5172560	1.2679488
2	23	2.066402	0.5309589	1.2665940
2	24	2.065437	0.5344523	1.2633582
2	25	2.079735	0.5316228	1.2667648
2	26	2.081797	0.5324254	1.2677712
2	27	2.081797	0.5324254	1.2677712
2	28	2.081797	0.5324254	1.2677712
2	29	2.081797	0.5324254	1.2677712
2	30	2.081797	0.5324254	1.2677712
2	31	2.081797	0.5324254	1.2677712
2	32	2.081797	0.5324254	1.2677712
2	33	2.081797	0.5324254	1.2677712
2	34	2.081797	0.5324254	1.2677712
2	35	2.081797	0.5324254	1.2677712
2	36	2.081797	0.5324254	1.2677712
2	37	2.081797	0.5324254	1.2677712
2	38	2.081797	0.5324254	1.2677712
3	2	1.407400	0.4656139	1.1174779
3	3	1.288493	0.5682931	1.0263117
3	4	1.379392	0.5892361	1.0481819
3	5	1.366985	0.5842811	1.0532546
3	6	1.363445	0.5710827	1.0519441
3	7	1.317884	0.6110113	0.9876128
3	8	2.155901	0.5337454	1.2717478
3	9	2.365837	0.5628087	1.3261618
3	10	1.345061	0.6164859	1.0209735
3	11	1.450749	0.5683842	1.0699845
3	12	1.436353	0.5685230	1.0624539
3	13	1.410479	0.5816416	1.0382527
3	14	1.715407	0.5519921	1.1549908
3	15	2.828992	0.4688554	1.4964577
3	16	2.831170	0.4265037	1.5243968
3	17	2.780085	0.4308703	1.5035084
3	18	2.807375	0.4281497	1.4931957
3	19	3.065332	0.4132803	1.6030211
3	20	3.042777	0.4181071	1.5828211
3	21	3.069110	0.4190439	1.5796859

```
  3       22      2.958982  0.4175394  1.5496401
  3       23      2.955995  0.4169054  1.5485249
  3       24      2.973481  0.4108560  1.5689330
  3       25      3.019521  0.3880575  1.5958254
  3       26      3.024177  0.3822188  1.5964017
  3       27      2.878881  0.3835998  1.5462256
  3       28      3.013091  0.3826830  1.5781993
  3       29      3.013091  0.3826830  1.5781993
  3       30      3.013091  0.3826830  1.5781993
  3       31      3.013091  0.3826830  1.5781993
  3       32      3.013091  0.3826830  1.5781993
  3       33      3.013091  0.3826830  1.5781993
  3       34      3.013091  0.3826830  1.5781993
  3       35      3.013091  0.3826830  1.5781993
  3       36      3.013091  0.3826830  1.5781993
  3       37      3.013091  0.3826830  1.5781993
  3       38      3.013091  0.3826830  1.5781993

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were nprune = 7 and degree = 2.
```

```r
mars_Pred2 <- predict(mars_model2, newdata = df_test)

## The function 'postResample' can be used to get the test set performance values
mars_pv2 <- postResample(pred = mars_Pred2, obs = df_test$Yield)

# model 5 - SVM - regression
set.seed(100)
svm_model2 <- train(Yield~., data = df_train,
 method = "svmRadial",
 preProc = c("center", "scale"),
 tuneLength = 14,
 trControl = trainControl(method="cv"))

#svm_model
svm_model2
```

```
Support Vector Machines with Radial Basis Function Kernel

144 samples
 56 predictor

Pre-processing: centered (56), scaled (56)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 129, 131, 128, 129, 129, 130, ...
Resampling results across tuning parameters:

  C        RMSE      Rsquared   MAE
     0.25  1.372028  0.5607359  1.1092751
     0.50  1.237001  0.6179908  1.0047103
     1.00  1.126298  0.6748623  0.9158092
     2.00  1.081110  0.6935348  0.8707261
     4.00  1.081172  0.6897600  0.8820878
     8.00  1.085143  0.6878698  0.8848315
    16.00  1.085143  0.6878698  0.8848315
```

```
    32.00  1.085143  0.6878698  0.8848315
    64.00  1.085143  0.6878698  0.8848315
   128.00  1.085143  0.6878698  0.8848315
   256.00  1.085143  0.6878698  0.8848315
   512.00  1.085143  0.6878698  0.8848315
  1024.00  1.085143  0.6878698  0.8848315
  2048.00  1.085143  0.6878698  0.8848315

Tuning parameter 'sigma' was held constant at a value of 0.01632049
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.01632049 and C = 2.
```

```r
svm_Pred2 <- predict(svm_model2, newdata = df_test)

## The function 'postResample' can be used to get the test set performance values
svm_pv2 <- postResample(pred = svm_Pred2, obs = df_test$Yield)
```

```r
# Model performance metrics
sum_t2 <- data.frame(
            knn_pv2,
            nnet_pv2,
            avnnet_pv2,
            mars_pv2,
            svm_pv2
            )

print(sum_t2)
```

```
           knn_pv2  nnet_pv2 avnnet_pv2  mars_pv2   svm_pv2
RMSE     1.4986649 1.6189836  1.6190778 1.4131453 1.2756876
Rsquared 0.3302198 0.3159727  0.3159386 0.3911259 0.5011139
MAE      1.2035764 1.3311308  1.3312383 1.0797807 1.0200509
```

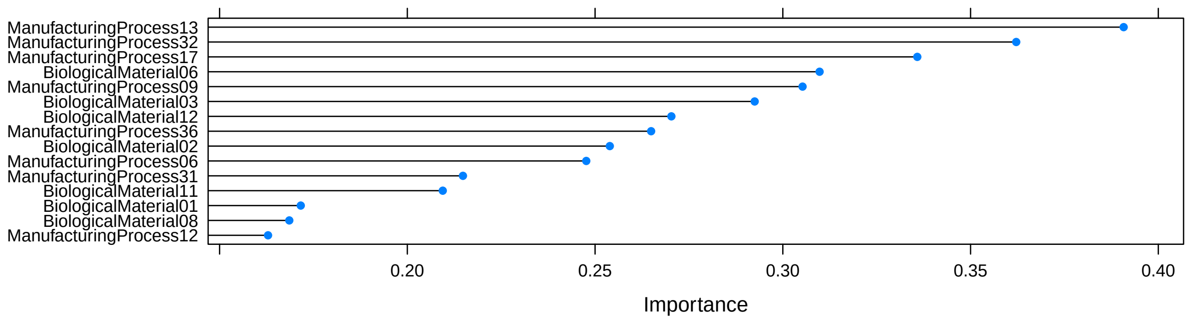
SVM regression gives the optimal performance based on RMSE, Rsquared and MAE on test set.

(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?

In linear model, `ManufacturingProcess32` was the most important predictor but in non-linear model, it is 2nd most important predictor - the most important predictor is `ManufacturingProcess13`.

In linear model, only 2 of top 10 were Biological where as in non-linear, 4 of them were.

```r
# code
varimp <- varImp(svm_model2,scale=F,useModel = T)
plot(varimp, top=15, scales = list(y = list(cex = 0.8)))
```



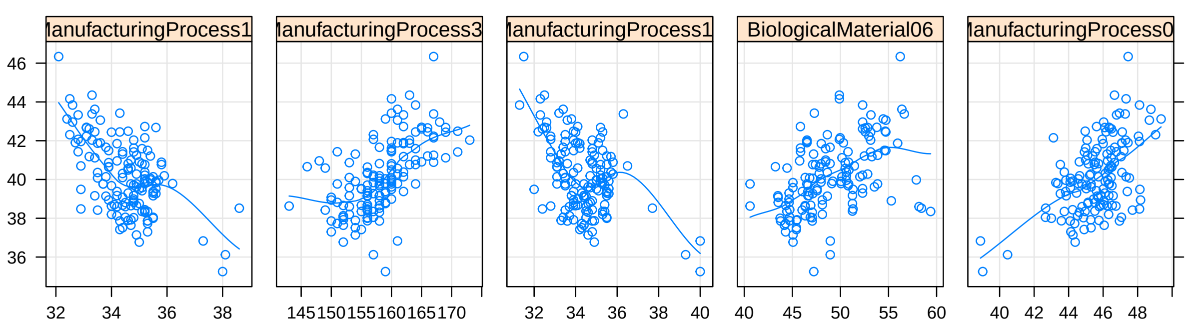
(c) Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

From Bivariate plot and correlation matrix, we know that `ManufacturingProcess32` has fairly positive relationship with `Yield` where as other 2 variables have fairly negative relationship. Among biological predictors, we know `BiologicalMaterial06` is the most important with fairly strong positive relationship with `Yield`.

This information can help researchers to focus more on `ManufacturingProcess32` and `BiologicalMaterial06` if their goal is to increase `Yield`.

```
# code
viporder <- order(abs(varimp$importance),decreasing=TRUE)
topVIP <- rownames(varimp$importance)[viporder[c(1:5)]]

# bivariate relationship
featurePlot(df_train[, topVIP],
            df_train$Yield,
            plot = "scatter",
            between = list(x = 1, y = 1),
            type = c("g", "p", "smooth"),
            layout = c(5,1),
            labels = rep("", 2))
```




```
# corr_matrix
corr_top5 <- cor(df_train[, topVIP], df_train$Yield, method = 'pearson', use = 'pairwise.complete.obs')
corr_top5
```

```
                           [,1]
```

```
ManufacturingProcess13 -0.5645290
ManufacturingProcess32  0.6017957
ManufacturingProcess17 -0.4666781
BiologicalMaterial06    0.4673292
ManufacturingProcess09  0.5423679
```