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')
```

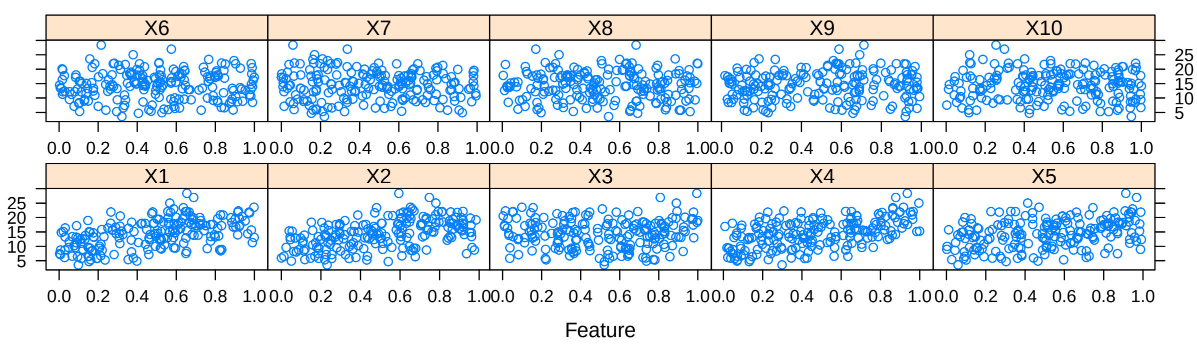
(1) Kuhn & Johnson 7.2

> 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)
trainingData$x <- data.frame(trainingData$x)

featurePlot(trainingData$x, trainingData$y)
```




```
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)
```

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

Model 1: KNN model with hyperparameter tuning

Train set CV performance - Hyperparameter tuning:

```
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.
```

Test set performance values:

```
     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

```
integer(0)
```

Train set CV performance - Hyperparameter tuning:

```
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.
```

Test set performance values:

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

We removed highly correlated predictors to ensure that the maximum absolute pariwise correlation between the predictors is less than 0.75. we did not have any highly correlated predictors so let's keep the features as they are.

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

Train set CV performance - Hyperparameter tuning:

```
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.
```

Test set performance values:

```
     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 (MARS)

Train set CV performance - Hyperparameter tuning:

```
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
```

Test set performance values:

```
     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

Train set CV performance - Hyperparameter tuning:

```
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.
```

Test set performance values:

```
     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.

	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

```
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?

Test set performance values:

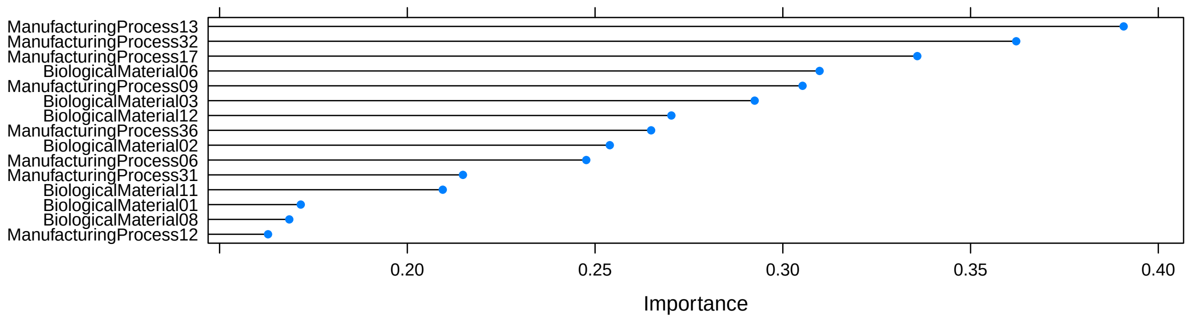
	knn_pv2	nnet_pv2	avnnet_pv2	mars_pv2	svm_pv2
RMSE	1.4986649	1.6189836	1.6190778	1.4131453	1.2756876
Rsquared	0.3302198	0.3159727	0.3159386	0.3911259	0.5011139
MAE	1.2035764	1.3311308	1.3312383	1.0797807	1.0200509

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`.

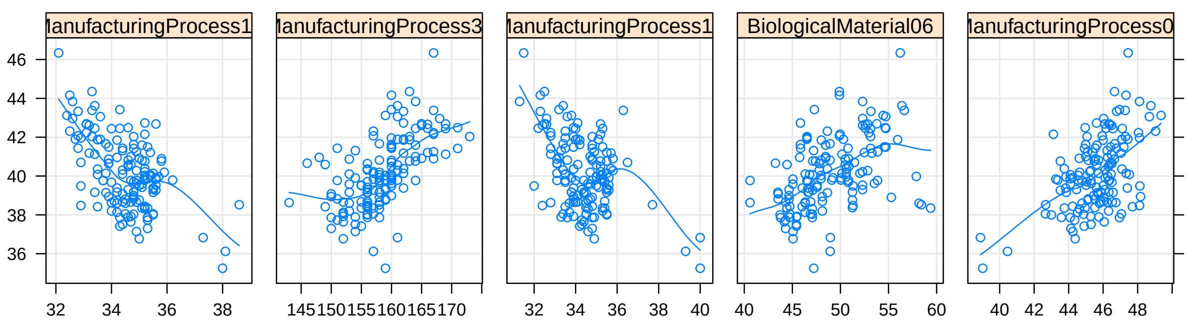
In linear model, only 2 of top 10 were Biological where as in non-linear, 4 of them were.



(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`.




```
                      corr_top5
ManufacturingProcess13 -0.5645290
ManufacturingProcess32  0.6017957
ManufacturingProcess17 -0.4666781
BiologicalMaterial06    0.4673292
ManufacturingProcess09  0.5423679
```

R Code

```
#insert all code here
# (7.2a)
##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
                   preProc = c("center", "scale"),
                   tuneGrid = data.frame(.k = 1:50),
                   trControl = trainControl(method = "cv"))

##Test set performance values:##
knn_Pred <- predict(knn_model, newdata = testData$x)
knn_pv <- postResample(pred = knn_Pred, obs = testData$y)

##Model 2: Neural Networks##

#findCorrelation(cor(trainingData$x), cutoff = .75)

# 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)

##Test set performance values:##
nnet_Pred <- predict(nnet_model, newdata = testData$x)
nnet_pv <- postResample(pred = nnet_Pred, obs = testData$y)

##Model 3: Neural Networks Using Model Averaging##
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"),
preProc = c("center", "scale"),
linout = TRUE,
trace = FALSE,
MaxNWts = 10 # (ncol(trainingData$x) + 1)  + 10 +  1,
maxit = 500)

##Test set performance values:##
avnnet_Pred <- predict(avnnet_model, newdata = testData$x)
avnnet_pv <- postResample(pred = avnnet_Pred, obs = testData$y)

##Model 4: Multivariate Adaptive Regression Splines (MARS)##
marsGrid <- expand.grid(.degree = 1:3, .nprune = 2:38)
```

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

##Test set performance values:##
mars_Pred <- predict(mars_model, newdata = testData$x)
mars_pv <- postResample(pred = mars_Pred, obs = testData$y)

##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"))

##Test set performance values:##
svm_Pred <- predict(svm_model, newdata = testData$x)
svm_pv <- postResample(pred = svm_Pred, obs = testData$y)

# (7.2b)
sum_t <- data.frame(
  knn_pv,
  nnet_pv,
  avnnet_pv,
  mars_pv,
  svm_pv
)

# (7.5a)
# Call code from 6.3
data("ChemicalManufacturingProcess")

# save df
df <- ChemicalManufacturingProcess

# set seed for split to allow for reproducibility
set.seed(20190227L)

# use mice w/ default settings to impute missing data
miceImput <- mice::mice(df, printFlag = FALSE)

# add imputed data to original data set
df_mice <- mice::complete(miceImput)

# Look for any features with no variance:
zero_cols <- nearZeroVar( df_mice )
df_final <- df_mice[,-zero_cols] # drop these zero variance columns

# 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_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_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_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"))

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_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)

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

```r
  svm_pv2
)

# (7.5b)
varimp <- varImp(svm_model2,scale=F,useModel = T)
#plot(varimp, top=15, scales = list(y = list(cex = 0.8)))

# (7.5c)
viporder <- order(abs(varimp$importance),decreasing=TRUE)
topVIP <- rownames(varimp$importance)[viporder[c(1:5)]]

#featurePlot(df_train[, topVIP],
#            df_train$Yield,
#            plot = "scatter",
#            between = list(x = 1, y = 1),
#            type = c("g", "p", "smooth"),
#            layout = c(5,1),
#            labels = rep("", 2))
corr_top5 <- cor(df_train[, topVIP], df_train$Yield, method = 'pearson', use = 'pairwise.complete.obs')
```