Data624 - Homework8

Amit Kapoor

4/19/2021

Contents

Exercise 7.2 **1**
Models . . . 3
Performance . . . 14

Exercise 7.5 **15**
(a) . . . 18
(b) . . . 30
(c) . . . 31

```
library(AppliedPredictiveModeling)
library(tidyverse)
library(caret)
library(mlbench)
library(naniar)
```

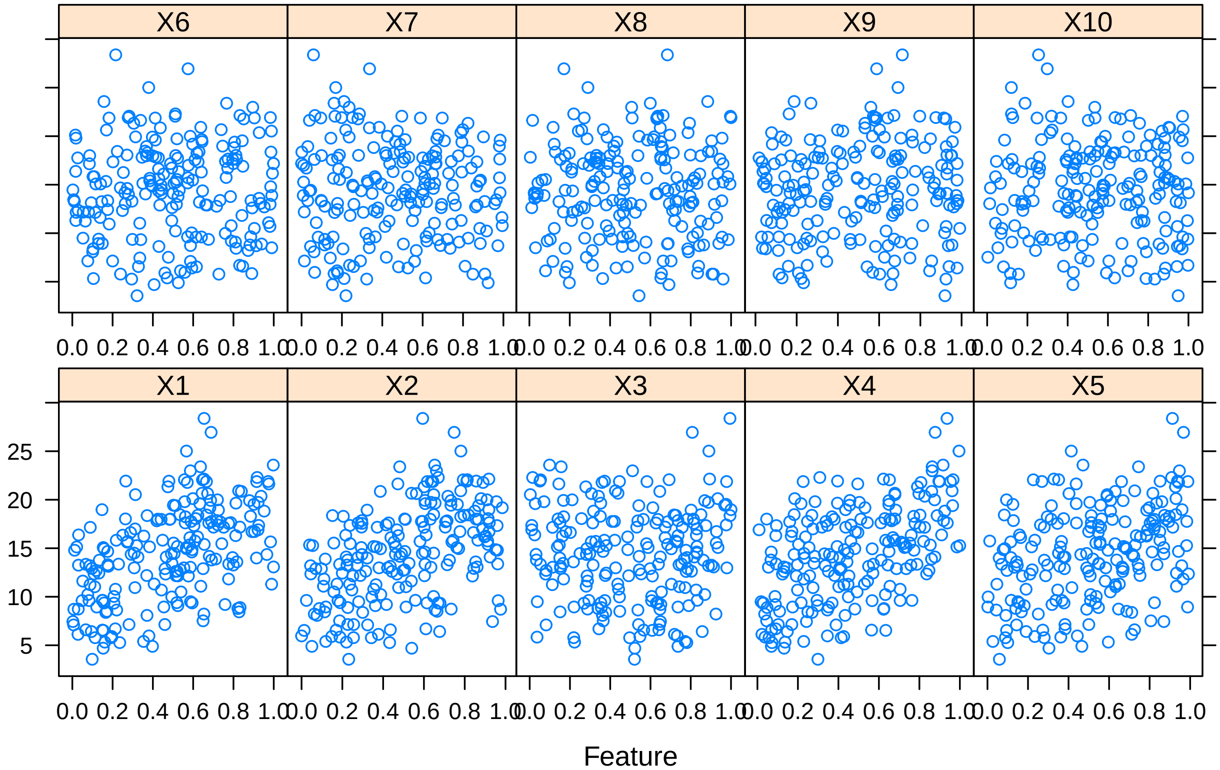
Exercise 7.2

Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data:

$$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + N(0, \sigma^2)$$

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)

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




```
glimpse(trainingData$x)
```

```
## Rows: 200
## Columns: 10
## $ X1  <dbl> 0.53377245, 0.58376503, 0.58957830, 0.69103989, 0.66733150, 0.8392~
## $ X2  <dbl> 0.64780643, 0.43815276, 0.58790649, 0.22595475, 0.81889851, 0.3862~
## $ X3  <dbl> 0.85078526, 0.67272659, 0.40967108, 0.03335447, 0.71676079, 0.6461~
## $ X4  <dbl> 0.181599574, 0.669249143, 0.338127280, 0.066912736, 0.803242873, 0~
## $ X5  <dbl> 0.929039760, 0.163797838, 0.894093335, 0.637445191, 0.083068641, 0~
## $ X6  <dbl> 0.36179060, 0.45305931, 0.02681911, 0.52500637, 0.22344157, 0.4370~
## $ X7  <dbl> 0.826660859, 0.648960076, 0.178561450, 0.513361395, 0.664490604, 0~
## $ X8  <dbl> 0.42140806, 0.84462393, 0.34959078, 0.79702598, 0.90389194, 0.6489~
## $ X9  <dbl> 0.59111440, 0.92819306, 0.01759542, 0.68986918, 0.39696995, 0.5311~
## $ X10 <dbl> 0.588621560, 0.758400814, 0.444118458, 0.445071622, 0.550080800, 0~
```

```
## 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)

glimpse(testData)
```

```
## List of 2
##  $ x:'data.frame':   5000 obs. of  10 variables:
##   ..$ X1 : num [1:5000] 0.4958 0.4078 0.4991 0.1956 0.0228 ...
##   ..$ X2 : num [1:5000] 0.261 0.716 0.715 0.369 0.746 ...
##   ..$ X3 : num [1:5000] 0.81 0.964 0.681 0.378 0.391 ...
##   ..$ X4 : num [1:5000] 0.82318 0.50565 0.00384 0.38569 0.87398 ...
##   ..$ X5 : num [1:5000] 0.822 0.88 0.498 0.279 0.197 ...
```

```
##    ..$ X6 : num [1:5000] 0.3219 0.5745 0.0603 0.5547 0.1762 ...
##    ..$ X7 : num [1:5000] 0.0544 0.4552 0.8926 0.3972 0.5067 ...
##    ..$ X8 : num [1:5000] 0.519 0.981 0.975 0.84 0.556 ...
##    ..$ X9 : num [1:5000] 0.3914 0.6663 0.0856 0.0904 0.379 ...
##    ..$ X10: num [1:5000] 0.73894 0.00059 0.59221 0.16227 0.65009 ...
##  $ y: num [1:5000] 17.52 20.87 12.82 5.09 10.79 ...
```

Models

Tune several models on these data.

K-Nearest Neighbors

The KNN algorithm assumes that similar things exist in close proximity. In other words, kNN approach simply predicts a new sample using the K-closest samples from the training set. Here we will use training using knn method on training data and find the besttune k value.

```r
set.seed(317)
knnfit <- train(trainingData$x,
                trainingData$y,
                method = "knn",
                preProcess = c("center","scale"),
                tuneLength = 20,
                trControl = trainControl(method = "cv"))

knnfit
```

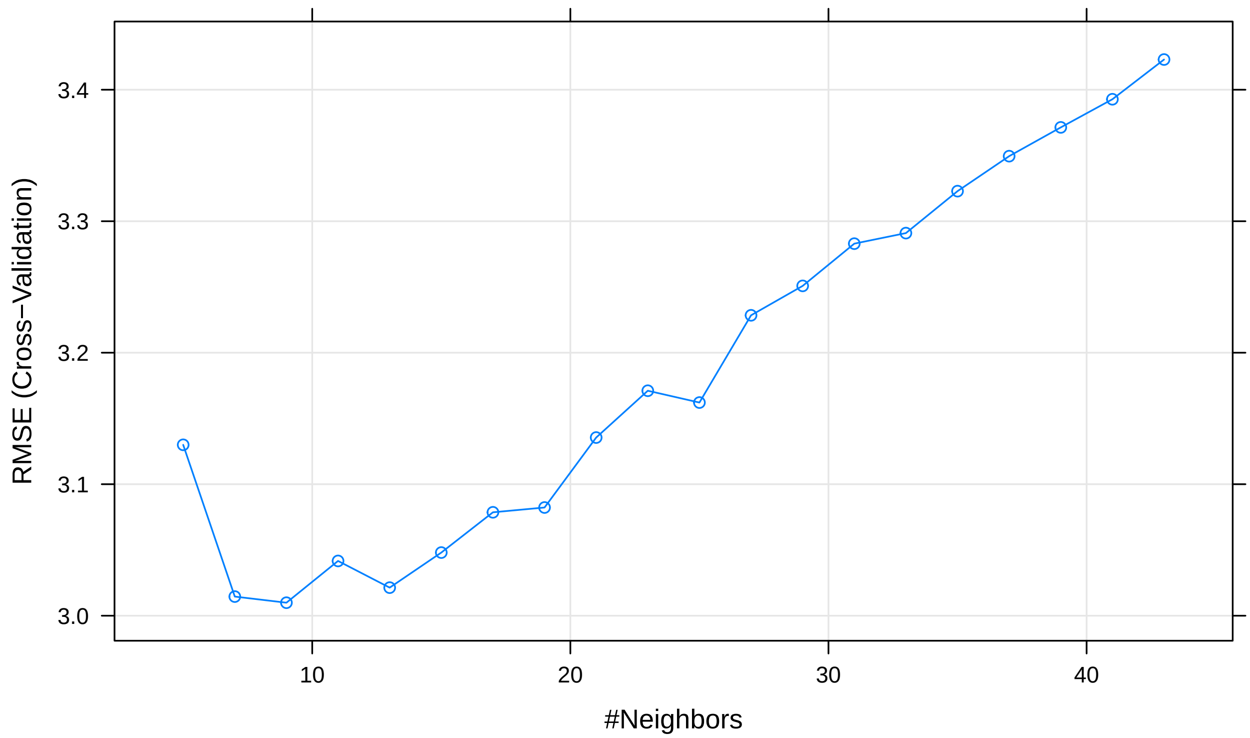
```
## k-Nearest Neighbors
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  3.129963  0.6307340  2.630432
##    7  3.014544  0.6725219  2.474808
##    9  3.009891  0.6866916  2.436532
##   11  3.041647  0.6922912  2.469379
##   13  3.021349  0.7218794  2.453776
##   15  3.048021  0.7287693  2.472147
##   17  3.078646  0.7320769  2.503486
##   19  3.082277  0.7434342  2.505638
##   21  3.135492  0.7305293  2.567035
##   23  3.171086  0.7317535  2.603795
##   25  3.162112  0.7447415  2.602762
##   27  3.228442  0.7314150  2.656904
##   29  3.250834  0.7278217  2.675701
##   31  3.282933  0.7267271  2.688565
##   33  3.290970  0.7350442  2.698592
##   35  3.322869  0.7305981  2.717600
##   37  3.349474  0.7317697  2.717001
```

```
##   39  3.371359  0.7308501  2.737718
##   41  3.392752  0.7396462  2.756124
##   43  3.422955  0.7437136  2.784268
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 9.
```

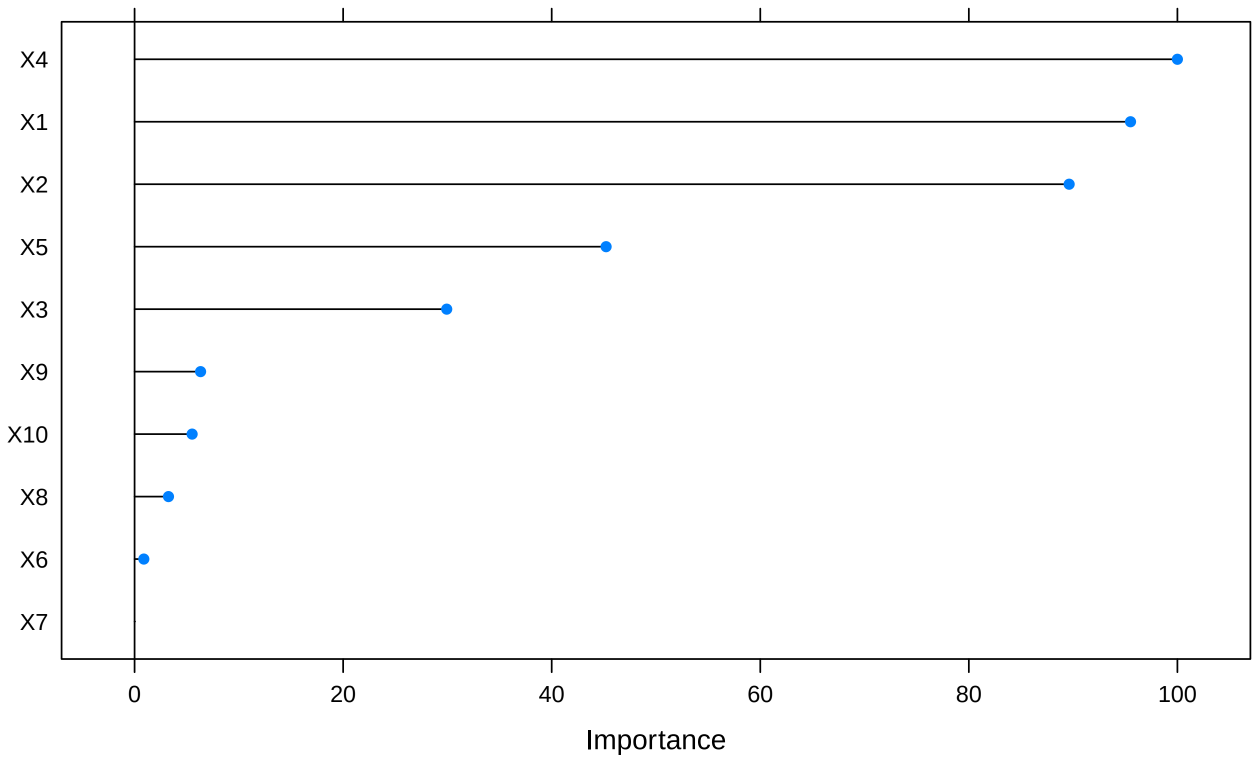
```r
# final parameters
knnfit$bestTune
```

```
##   k
## 3 9
```

```r
# plot RMSE
plot(knnfit)
```




```r
# plot variable importance
plot(varImp(knnfit))
```




```
data.frame(Rsquared=knnfit[["results"]][["Rsquared"]][as.numeric(rownames(knnfit$bestTune))],
           RMSE=knnfit[["results"]][["RMSE"]][as.numeric(rownames(knnfit$bestTune))])
```

```
##    Rsquared     RMSE
## 1 0.6866916 3.009891
```

It is evident here that the best value of K is 9 which resulted Rsquared as 0.69 and RMSE as 3.01. Also the top 5 top predictors are X4, X1, X2, X5 and X3.

Support Vector Machines

The objective of the support vector machine algo is to find a hyperplane in an N-dimensional space (N being the number of features) that classifies the data points. Here we will use training using svmRadial method .

```
set.seed(317)
svmfit <- train(trainingData$x,
                trainingData$y,
                method = "svmRadial",
                preProcess = c("center","scale"),
                tuneLength = 20,
                trControl = trainControl(method = "cv"))

svmfit
```

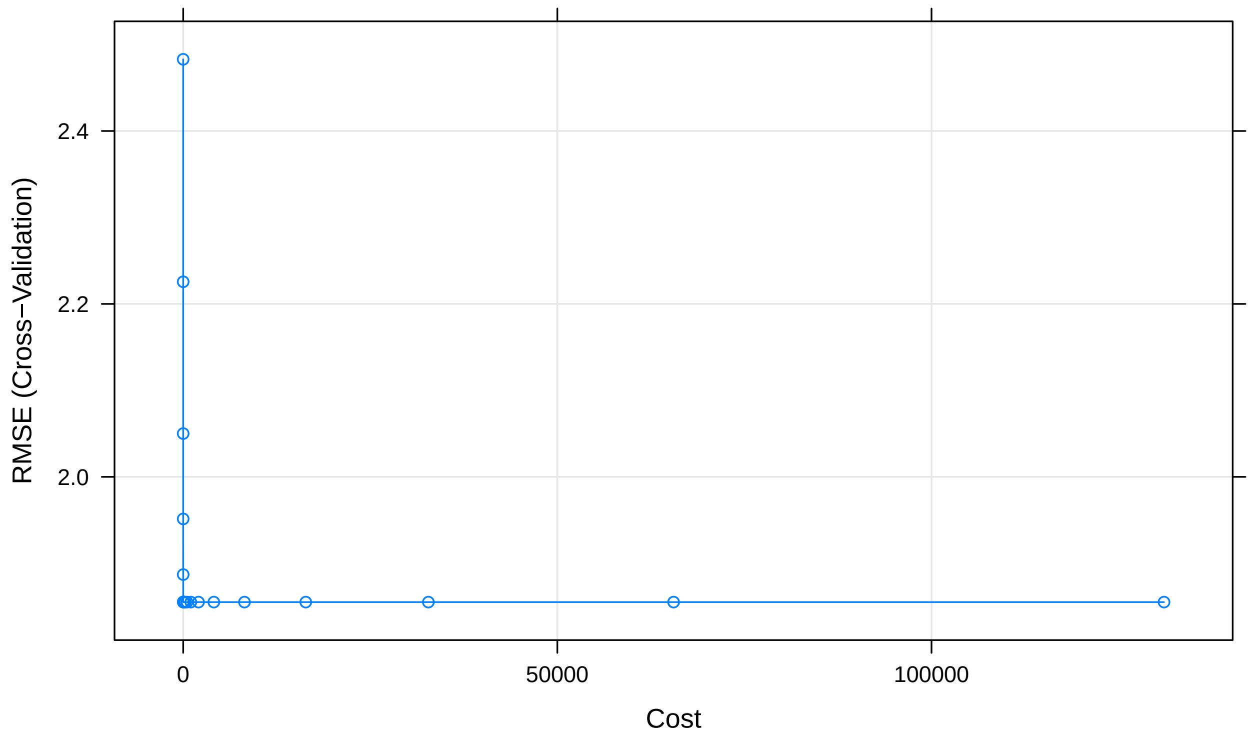
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
```

```
## Resampling results across tuning parameters:
##
##   C          RMSE      Rsquared   MAE
##        0.25  2.482921  0.8041684  1.987997
##        0.50  2.225629  0.8199832  1.770148
##        1.00  2.050095  0.8408862  1.642206
##        2.00  1.951377  0.8548928  1.553396
##        4.00  1.887021  0.8632458  1.502009
##        8.00  1.855350  0.8658251  1.469802
##       16.00  1.855273  0.8652878  1.471794
##       32.00  1.855180  0.8652888  1.471609
##       64.00  1.855180  0.8652888  1.471609
##      128.00  1.855180  0.8652888  1.471609
##      256.00  1.855180  0.8652888  1.471609
##      512.00  1.855180  0.8652888  1.471609
##     1024.00  1.855180  0.8652888  1.471609
##     2048.00  1.855180  0.8652888  1.471609
##     4096.00  1.855180  0.8652888  1.471609
##     8192.00  1.855180  0.8652888  1.471609
##    16384.00  1.855180  0.8652888  1.471609
##    32768.00  1.855180  0.8652888  1.471609
##    65536.00  1.855180  0.8652888  1.471609
##   131072.00  1.855180  0.8652888  1.471609
##
## Tuning parameter 'sigma' was held constant at a value of 0.06295544
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.06295544 and C = 32.
```

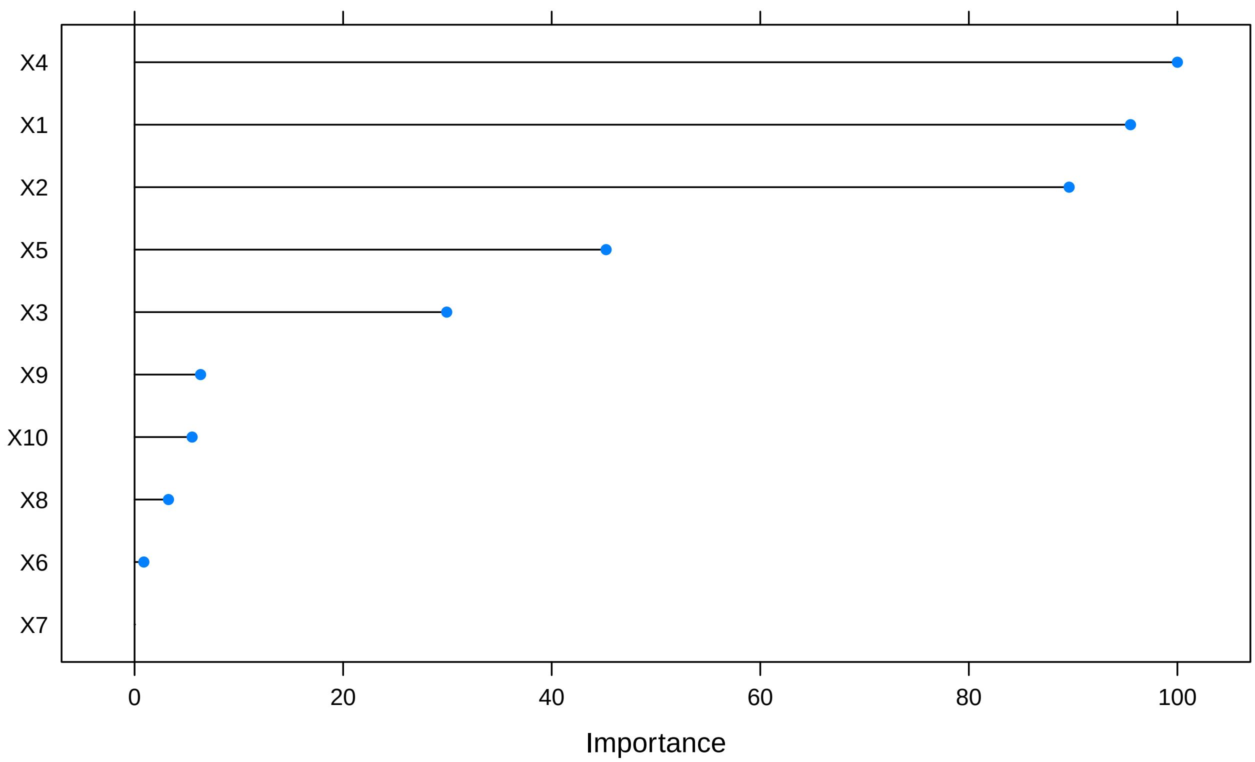
```
svmfit$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 32
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.062955443796397
##
## Number of Support Vectors : 152
##
## Objective Function Value : -73.5893
## Training error : 0.0085
```

```
# plot RMSE
plot(svmfit)
```




```r
# plot variable importance
plot(varImp(svmfit))
```




```r
data.frame(Rsquared=svmfit[["results"]][["Rsquared"]][as.numeric(rownames(svmfit$bestTune))],
           RMSE=svmfit[["results"]][["RMSE"]][as.numeric(rownames(svmfit$bestTune))])
```

```
##    Rsquared    RMSE
## 1 0.8652888 1.85518
```

So we can see here that best SVM model produced Rsquared as 0.87 and RMSE as 1.86. Tuning parameter 'sigma' was held constant at a value of 0.063 RMSE was used to select the optimal model using the smallest value. Also the top 5 top predictors are X4, X1, X2, X5 and X3.

Multivariate Adaptive Regression Splines

MARS creates a piecewise linear model which provides an intuitive stepping block into non-linearity after grasping the concept of multiple linear regression. MARS provided a convenient approach to capture the nonlinear relationships in the data by assessing cutpoints (knots) similar to step functions. The procedure assesses each data point for each predictor as a knot and creates a linear regression model with the candidate features.

```
set.seed(317)
marsGrid <- expand.grid(.degree=1:2, .nprune=2:38)
marsfit <- train(trainingData$x,
                trainingData$y,
                method = "earth",
                preProcess = c("center","scale"),
                tuneGrid = marsGrid,
                trControl = trainControl(method = "cv"))
```

```
## Loading required package: earth

## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos
```

```
marsfit
```

```
## Multivariate Adaptive Regression Spline
##
## 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:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      4.425366  0.2190557  3.6620782
##   1        3      3.510669  0.5027292  2.8172393
##   1        4      2.659861  0.7244814  2.1491495
##   1        5      2.357542  0.7748479  1.8846523
##   1        6      2.267014  0.7950771  1.8032647
##   1        7      1.747556  0.8845023  1.3957204
##   1        8      1.742217  0.8839879  1.3446484
##   1        9      1.686370  0.8895096  1.2940316
##   1       10      1.611802  0.9000011  1.2485375
##   1       11      1.621181  0.8968899  1.2597303
##   1       12      1.608874  0.8973276  1.2577114
##   1       13      1.598875  0.8990619  1.2451770
```

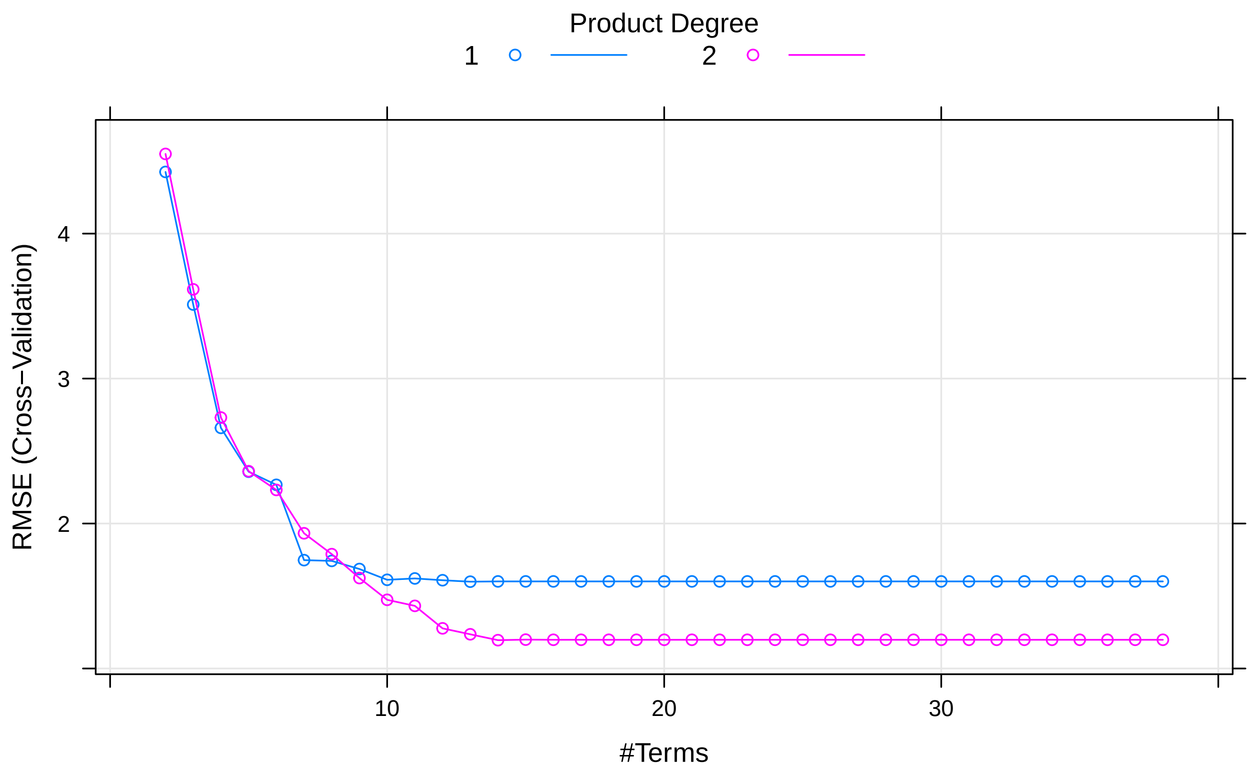
```
## 1 14 1.600854 0.8985110 1.2482796
## 1 15 1.600854 0.8985110 1.2482796
## 1 16 1.600854 0.8985110 1.2482796
## 1 17 1.600854 0.8985110 1.2482796
## 1 18 1.600854 0.8985110 1.2482796
## 1 19 1.600854 0.8985110 1.2482796
## 1 20 1.600854 0.8985110 1.2482796
## 1 21 1.600854 0.8985110 1.2482796
## 1 22 1.600854 0.8985110 1.2482796
## 1 23 1.600854 0.8985110 1.2482796
## 1 24 1.600854 0.8985110 1.2482796
## 1 25 1.600854 0.8985110 1.2482796
## 1 26 1.600854 0.8985110 1.2482796
## 1 27 1.600854 0.8985110 1.2482796
## 1 28 1.600854 0.8985110 1.2482796
## 1 29 1.600854 0.8985110 1.2482796
## 1 30 1.600854 0.8985110 1.2482796
## 1 31 1.600854 0.8985110 1.2482796
## 1 32 1.600854 0.8985110 1.2482796
## 1 33 1.600854 0.8985110 1.2482796
## 1 34 1.600854 0.8985110 1.2482796
## 1 35 1.600854 0.8985110 1.2482796
## 1 36 1.600854 0.8985110 1.2482796
## 1 37 1.600854 0.8985110 1.2482796
## 1 38 1.600854 0.8985110 1.2482796
## 2  2 4.549565 0.1746915 3.7544582
## 2  3 3.615256 0.4741270 2.9301983
## 2  4 2.731108 0.7057270 2.1797808
## 2  5 2.361050 0.7739228 1.8736496
## 2  6 2.231880 0.8022071 1.7443082
## 2  7 1.932782 0.8498407 1.5459941
## 2  8 1.788846 0.8794599 1.3858674
## 2  9 1.623900 0.9014211 1.2410832
## 2 10 1.473741 0.9171042 1.1762413
## 2 11 1.432077 0.9268157 1.1481451
## 2 12 1.276945 0.9409982 1.0218556
## 2 13 1.235949 0.9430223 0.9945005
## 2 14 1.195378 0.9473300 0.9628314
## 2 15 1.199243 0.9471786 0.9611487
## 2 16 1.198156 0.9471995 0.9701514
## 2 17 1.198156 0.9471995 0.9701514
## 2 18 1.198156 0.9471995 0.9701514
## 2 19 1.198156 0.9471995 0.9701514
## 2 20 1.198156 0.9471995 0.9701514
## 2 21 1.198156 0.9471995 0.9701514
## 2 22 1.198156 0.9471995 0.9701514
## 2 23 1.198156 0.9471995 0.9701514
## 2 24 1.198156 0.9471995 0.9701514
## 2 25 1.198156 0.9471995 0.9701514
## 2 26 1.198156 0.9471995 0.9701514
## 2 27 1.198156 0.9471995 0.9701514
## 2 28 1.198156 0.9471995 0.9701514
## 2 29 1.198156 0.9471995 0.9701514
## 2 30 1.198156 0.9471995 0.9701514
```

```
##   2        31       1.198156  0.9471995  0.9701514
##   2        32       1.198156  0.9471995  0.9701514
##   2        33       1.198156  0.9471995  0.9701514
##   2        34       1.198156  0.9471995  0.9701514
##   2        35       1.198156  0.9471995  0.9701514
##   2        36       1.198156  0.9471995  0.9701514
##   2        37       1.198156  0.9471995  0.9701514
##   2        38       1.198156  0.9471995  0.9701514
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 14 and degree = 2.
```

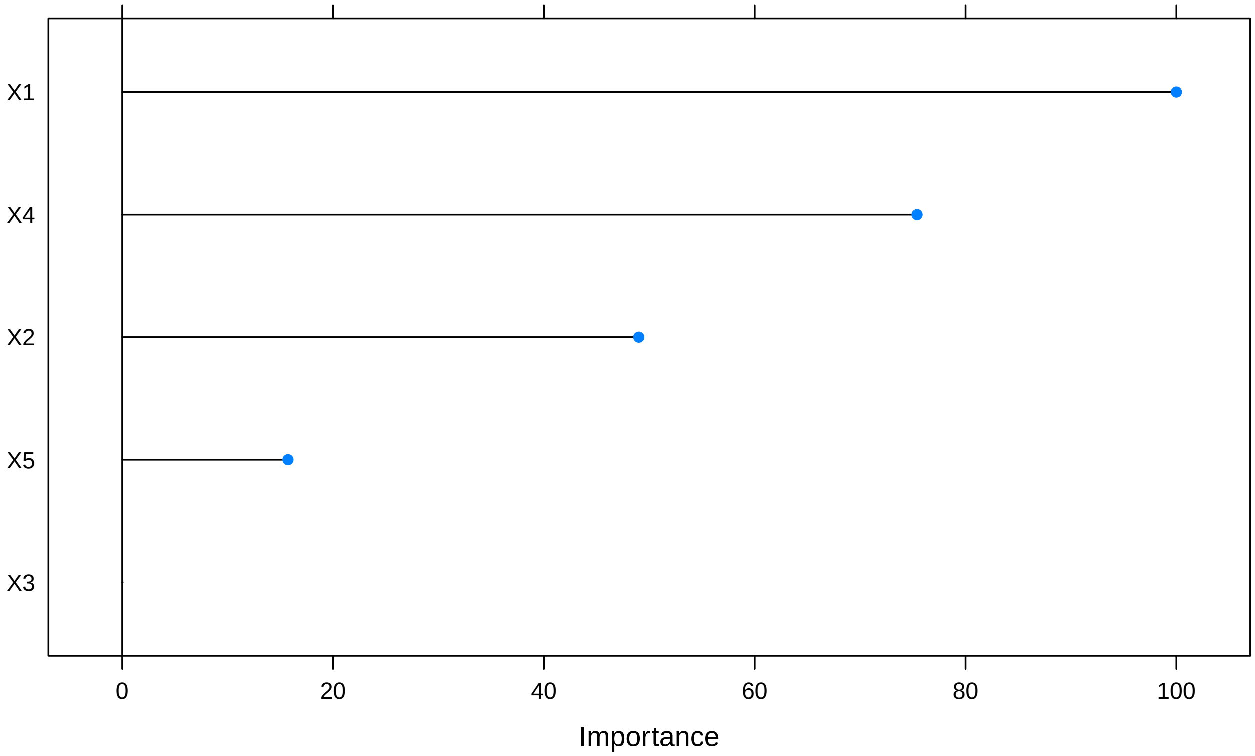
```
# final parameters
marsfit$bestTune
```

```
##    nprune degree
## 50     14      2
```

```
# plot RMSE
plot(marsfit)
```




```
# plot variable importance
plot(varImp(marsfit))
```




```
data.frame(Rsquared=marsfit[["results"]][["Rsquared"]][as.numeric(rownames(marsfit$bestTune))],
           RMSE=marsfit[["results"]][["RMSE"]][as.numeric(rownames(marsfit$bestTune))])
```

```
##    Rsquared     RMSE
## 1 0.9471995 1.198156
```

RMSE was used to select the optimal model using the smallest value. The final values used for the model were nprune = 14 and degree = 2 that resulted Rsquared as 0.94 and RMSE as 1.20. So far we can see MARS model has a best fit on training data comparing with KNN and SVM. Also the top predictors are X1, X4, X2 and X5.

Neural Networks

Neural Networks are nonlinear regression techniques inspired by theories about how the brain works. The outcome is modeled by an intermediary set of unobserved variables (hidden variables). These hidden units are linear combinations of the original predictors.

```
set.seed(317)
nnetGrid <- expand.grid(.decay=c(0,0.01,.1),
                        .size=c(1:10),
                        .bag=FALSE)

nnetfit <- train(trainingData$x,
                 trainingData$y,
                 method = "avNNet",
                 tuneGrid = nnetGrid,
                 preProcess = c("center","scale"),
                 linout = TRUE,
                 trace = FALSE,
                 MaxNWts =10 * (ncol(trainingData$x)+1) +10+1,
                 maxit=500)
```

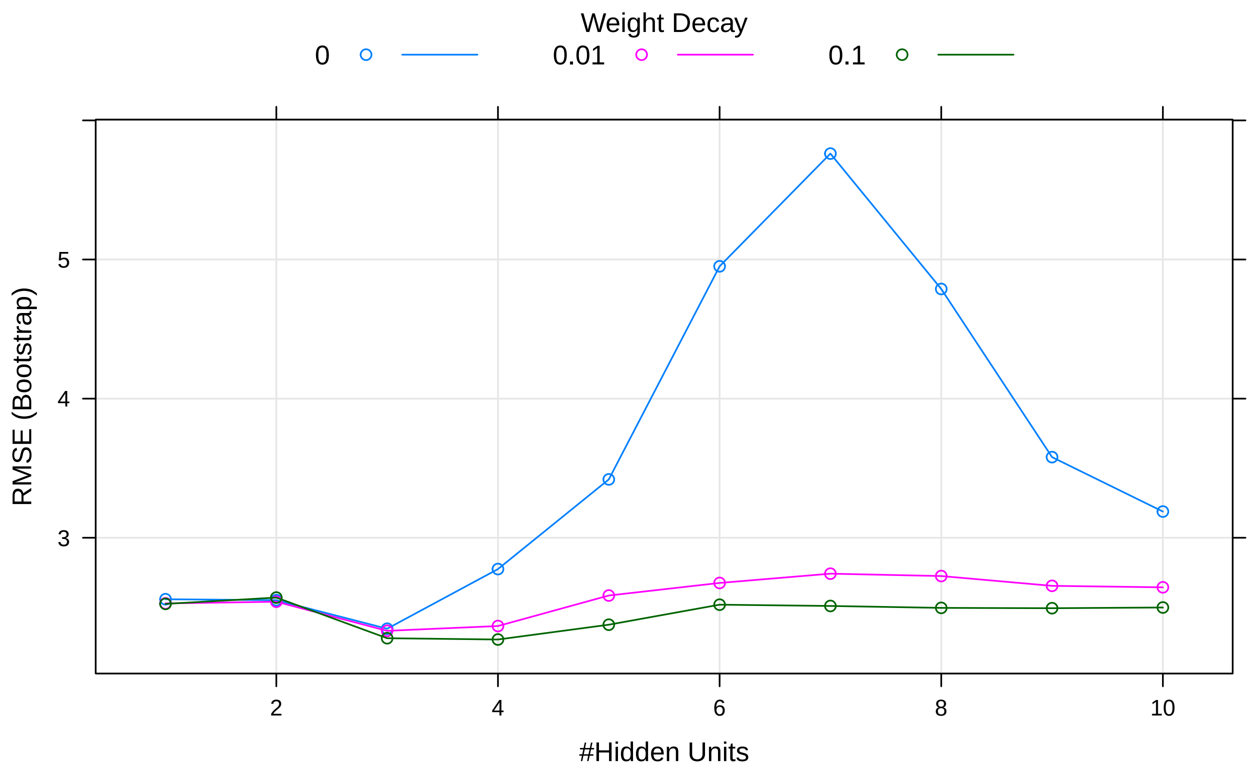
```r
nnetfit
```

```
## 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    2.558074  0.7354749  2.018750
##   0.00    2    2.551324  0.7322727  2.006227
##   0.00    3    2.346114  0.7746725  1.844363
##   0.00    4    2.774983  0.7050149  2.101071
##   0.00    5    3.419444  0.6062167  2.415627
##   0.00    6    4.951063  0.4876851  3.177594
##   0.00    7    5.761377  0.4080602  3.761079
##   0.00    8    4.788191  0.4487625  3.173436
##   0.00    9    3.579533  0.6186456  2.583179
##   0.00   10    3.188433  0.6596291  2.358802
##   0.01    1    2.528201  0.7392076  1.977816
##   0.01    2    2.540150  0.7338716  2.007651
##   0.01    3    2.331119  0.7736264  1.837698
##   0.01    4    2.365476  0.7719779  1.865425
##   0.01    5    2.584746  0.7330244  2.031432
##   0.01    6    2.675065  0.7208631  2.135200
##   0.01    7    2.741729  0.7094062  2.182309
##   0.01    8    2.724735  0.7068107  2.131129
##   0.01    9    2.654345  0.7162791  2.140507
##   0.01   10    2.643604  0.7185392  2.106341
##   0.10    1    2.524669  0.7388254  1.975027
##   0.10    2    2.570081  0.7270239  2.014844
##   0.10    3    2.277826  0.7854825  1.801937
##   0.10    4    2.268553  0.7881324  1.809673
##   0.10    5    2.374965  0.7694193  1.883229
##   0.10    6    2.518906  0.7442645  1.988500
##   0.10    7    2.509753  0.7472883  1.995038
##   0.10    8    2.495911  0.7495486  1.971962
##   0.10    9    2.493696  0.7469856  1.982746
##   0.10   10    2.498591  0.7449700  1.991270
##
## 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.
```

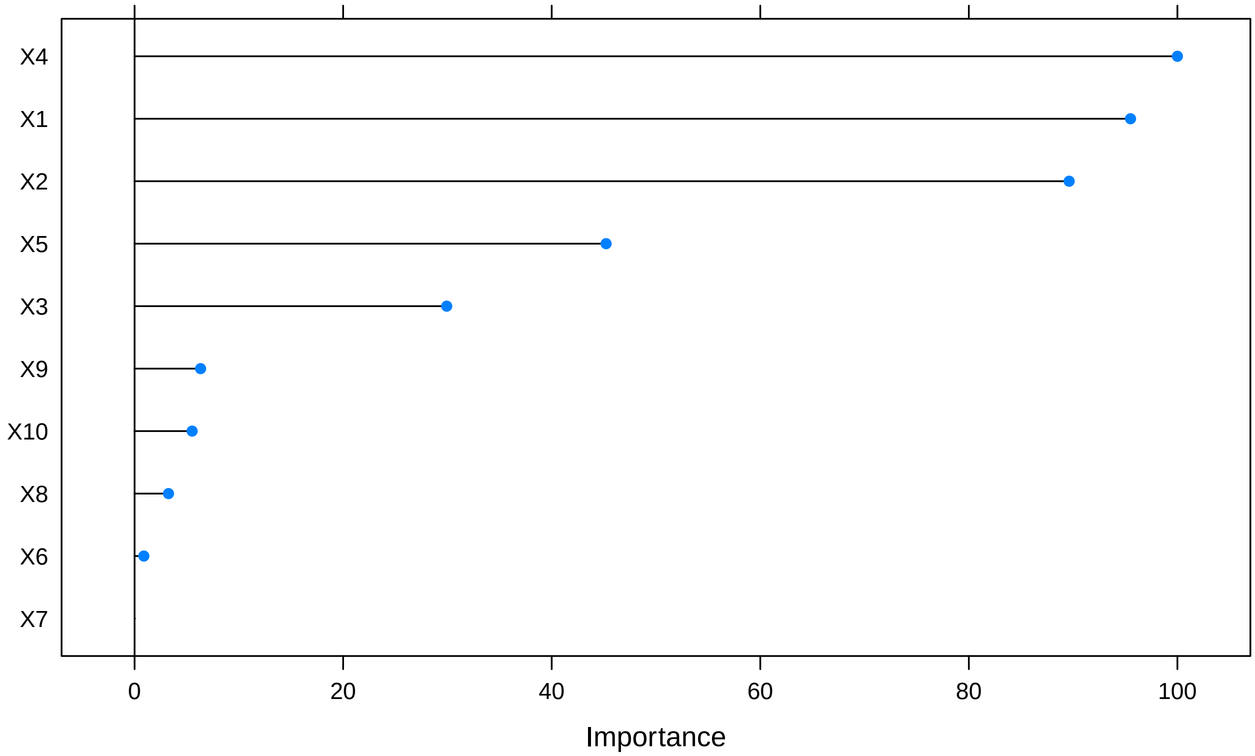
```r
# final parameters
nnetfit$bestTune
```

```
##    size decay   bag
## 24    4   0.1 FALSE
```

```
# plot RMSE
plot(nnetfit)
```




```
# plot variable importance
plot(varImp(nnetfit))
```




```r
data.frame(Rsquared=nnetfit[["results"]][["Rsquared"]][as.numeric(rownames(nnetfit$bestTune))],
           RMSE=nnetfit[["results"]][["RMSE"]][as.numeric(rownames(nnetfit$bestTune))])
```

```
##    Rsquared     RMSE
## 1 0.7495486 2.495911
```

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 that resulted the Rsquared 0.75 and RMSE as 2.50. The top predictors come up are X4, X1, X2, X5 and X3.

Performance

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

```r
set.seed(317)
knn.pred <- predict(knnfit, newdata = testData$x)
svm.pred <- predict(svmfit, newdata = testData$x)
mars.pred <- predict(marsfit, newdata = testData$x)
nnet.pred <- predict(nnetfit, newdata = testData$x)


data.frame(rbind(KNN=postResample(pred=knn.pred,obs = testData$y),
                 SVM=postResample(pred=svm.pred,obs = testData$y),
                 MARS=postResample(pred=mars.pred,obs = testData$y),
                 NNET=postResample(pred=nnet.pred,obs = testData$y)))
```

```
##          RMSE  Rsquared      MAE
## KNN  3.117232 0.6556622 2.489991
## SVM  2.073617 0.8256703 1.575110
## MARS 1.277999 0.9338365 1.014707
```

```
## NNET 2.162285 0.8168289 1.615305
```

From the results, it is evident that the best model is MARS with $R^2 = 0.93$ and min RMSE = 1.28 on test data. The MARS does select the informative predictors X1-X5.

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

```
data(ChemicalManufacturingProcess)
```

```
glimpse(ChemicalManufacturingProcess)
```

```
## Rows: 176
## Columns: 58
## $ Yield                  <dbl> 38.00, 42.44, 42.03, 41.42, 42.49, 43.57, 43.12~
## $ BiologicalMaterial01   <dbl> 6.25, 8.01, 8.01, 8.01, 7.47, 6.12, 7.48, 6.94,~
## $ BiologicalMaterial02   <dbl> 49.58, 60.97, 60.97, 60.97, 63.33, 58.36, 64.47~
## $ BiologicalMaterial03   <dbl> 56.97, 67.48, 67.48, 67.48, 72.25, 65.31, 72.41~
## $ BiologicalMaterial04   <dbl> 12.74, 14.65, 14.65, 14.65, 14.02, 15.17, 13.82~
## $ BiologicalMaterial05   <dbl> 19.51, 19.36, 19.36, 19.36, 17.91, 21.79, 17.71~
## $ BiologicalMaterial06   <dbl> 43.73, 53.14, 53.14, 53.14, 54.66, 51.23, 54.45~
## $ BiologicalMaterial07   <dbl> 100, 100, 100, 100, 100, 100, 100, 100, 100, 10~
## $ BiologicalMaterial08   <dbl> 16.66, 19.04, 19.04, 19.04, 18.22, 18.30, 18.72~
## $ BiologicalMaterial09   <dbl> 11.44, 12.55, 12.55, 12.55, 12.80, 12.13, 12.95~
## $ BiologicalMaterial10   <dbl> 3.46, 3.46, 3.46, 3.46, 3.05, 3.78, 3.04, 3.85,~
## $ BiologicalMaterial11   <dbl> 138.09, 153.67, 153.67, 153.67, 147.61, 151.88,~
## $ BiologicalMaterial12   <dbl> 18.83, 21.05, 21.05, 21.05, 21.05, 20.76, 20.75~
## $ ManufacturingProcess01 <dbl> NA, 0.0, 0.0, 0.0, 10.7, 12.0, 11.5, 12.0, 12.0~
## $ ManufacturingProcess02 <dbl> NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ ManufacturingProcess03 <dbl> NA, NA, NA, NA, NA, NA, 1.56, 1.55, 1.56, 1.55,~
## $ ManufacturingProcess04 <dbl> NA, 917, 912, 911, 918, 924, 933, 929, 928, 938~
## $ ManufacturingProcess05 <dbl> NA, 1032.2, 1003.6, 1014.6, 1027.5, 1016.8, 988~
## $ ManufacturingProcess06 <dbl> NA, 210.0, 207.1, 213.3, 205.7, 208.9, 210.0, 2~
## $ ManufacturingProcess07 <dbl> NA, 177, 178, 177, 178, 178, 177, 178, 177, 177~
## $ ManufacturingProcess08 <dbl> NA, 178, 178, 177, 178, 178, 178, 178, 177, 177~
## $ ManufacturingProcess09 <dbl> 43.00, 46.57, 45.07, 44.92, 44.96, 45.32, 49.36~
## $ ManufacturingProcess10 <dbl> NA, NA, NA, NA, NA, NA, 11.6, 10.2, 9.7, 10.1, ~
## $ ManufacturingProcess11 <dbl> NA, NA, NA, NA, NA, NA, 11.5, 11.3, 11.1, 10.2,~
## $ ManufacturingProcess12 <dbl> NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ ManufacturingProcess13 <dbl> 35.5, 34.0, 34.8, 34.8, 34.6, 34.0, 32.4, 33.6,~
## $ ManufacturingProcess14 <dbl> 4898, 4869, 4878, 4897, 4992, 4985, 4745, 4854,~
## $ ManufacturingProcess15 <dbl> 6108, 6095, 6087, 6102, 6233, 6222, 5999, 6105,~
## $ ManufacturingProcess16 <dbl> 4682, 4617, 4617, 4635, 4733, 4786, 4486, 4626,~
## $ ManufacturingProcess17 <dbl> 35.5, 34.0, 34.8, 34.8, 33.9, 33.4, 33.8, 33.6,~
## $ ManufacturingProcess18 <dbl> 4865, 4867, 4877, 4872, 4886, 4862, 4758, 4766,~
## $ ManufacturingProcess19 <dbl> 6049, 6097, 6078, 6073, 6102, 6115, 6013, 6022,~
## $ ManufacturingProcess20 <dbl> 4665, 4621, 4621, 4611, 4659, 4696, 4522, 4552,~
## $ ManufacturingProcess21 <dbl> 0.0, 0.0, 0.0, 0.0, -0.7, -0.6, 1.4, 0.0, 0.0, ~
## $ ManufacturingProcess22 <dbl> NA, 3, 4, 5, 8, 9, 1, 2, 3, 4, 6, 7, 8, 10, 11,~
## $ ManufacturingProcess23 <dbl> NA, 0, 1, 2, 4, 1, 1, 2, 3, 1, 3, 4, 1, 2, 3, 4~
## $ ManufacturingProcess24 <dbl> NA, 3, 4, 5, 18, 1, 1, 2, 3, 4, 6, 7, 8, 2, 15,~
## $ ManufacturingProcess25 <dbl> 4873, 4869, 4897, 4892, 4930, 4871, 4795, 4806,~
## $ ManufacturingProcess26 <dbl> 6074, 6107, 6116, 6111, 6151, 6128, 6057, 6059,~
```

```
## $ ManufacturingProcess27 <dbl> 4685, 4630, 4637, 4630, 4684, 4687, 4572, 4586,~
## $ ManufacturingProcess28 <dbl> 10.7, 11.2, 11.1, 11.1, 11.3, 11.4, 11.2, 11.1,~
## $ ManufacturingProcess29 <dbl> 21.0, 21.4, 21.3, 21.3, 21.6, 21.7, 21.2, 21.2,~
## $ ManufacturingProcess30 <dbl> 9.9, 9.9, 9.4, 9.4, 9.0, 10.1, 11.2, 10.9, 10.5~
## $ ManufacturingProcess31 <dbl> 69.1, 68.7, 69.3, 69.3, 69.4, 68.2, 67.6, 67.9,~
## $ ManufacturingProcess32 <dbl> 156, 169, 173, 171, 171, 173, 159, 161, 160, 16~
## $ ManufacturingProcess33 <dbl> 66, 66, 66, 68, 70, 70, 65, 65, 65, 66, 67, 67,~
## $ ManufacturingProcess34 <dbl> 2.4, 2.6, 2.6, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.~
## $ ManufacturingProcess35 <dbl> 486, 508, 509, 496, 468, 490, 475, 478, 491, 48~
## $ ManufacturingProcess36 <dbl> 0.019, 0.019, 0.018, 0.018, 0.017, 0.018, 0.019~
## $ ManufacturingProcess37 <dbl> 0.5, 2.0, 0.7, 1.2, 0.2, 0.4, 0.8, 1.0, 1.2, 1.~
## $ ManufacturingProcess38 <dbl> 3, 2, 2, 2, 2, 2, 2, 2, 3, 3, 2, 3, 3, 3, 3, 3,~
## $ ManufacturingProcess39 <dbl> 7.2, 7.2, 7.2, 7.2, 7.3, 7.2, 7.3, 7.3, 7.4, 7.~
## $ ManufacturingProcess40 <dbl> NA, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0~
## $ ManufacturingProcess41 <dbl> NA, 0.15, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0~
## $ ManufacturingProcess42 <dbl> 11.6, 11.1, 12.0, 10.6, 11.0, 11.5, 11.7, 11.4,~
## $ ManufacturingProcess43 <dbl> 3.0, 0.9, 1.0, 1.1, 1.1, 2.2, 0.7, 0.8, 0.9, 0.~
## $ ManufacturingProcess44 <dbl> 1.8, 1.9, 1.8, 1.8, 1.7, 1.8, 2.0, 2.0, 1.9, 1.~
## $ ManufacturingProcess45 <dbl> 2.4, 2.2, 2.3, 2.1, 2.1, 2.0, 2.2, 2.2, 2.1, 2.~
```

The matrix processPredictors contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. yield contains the percent yield for each run.

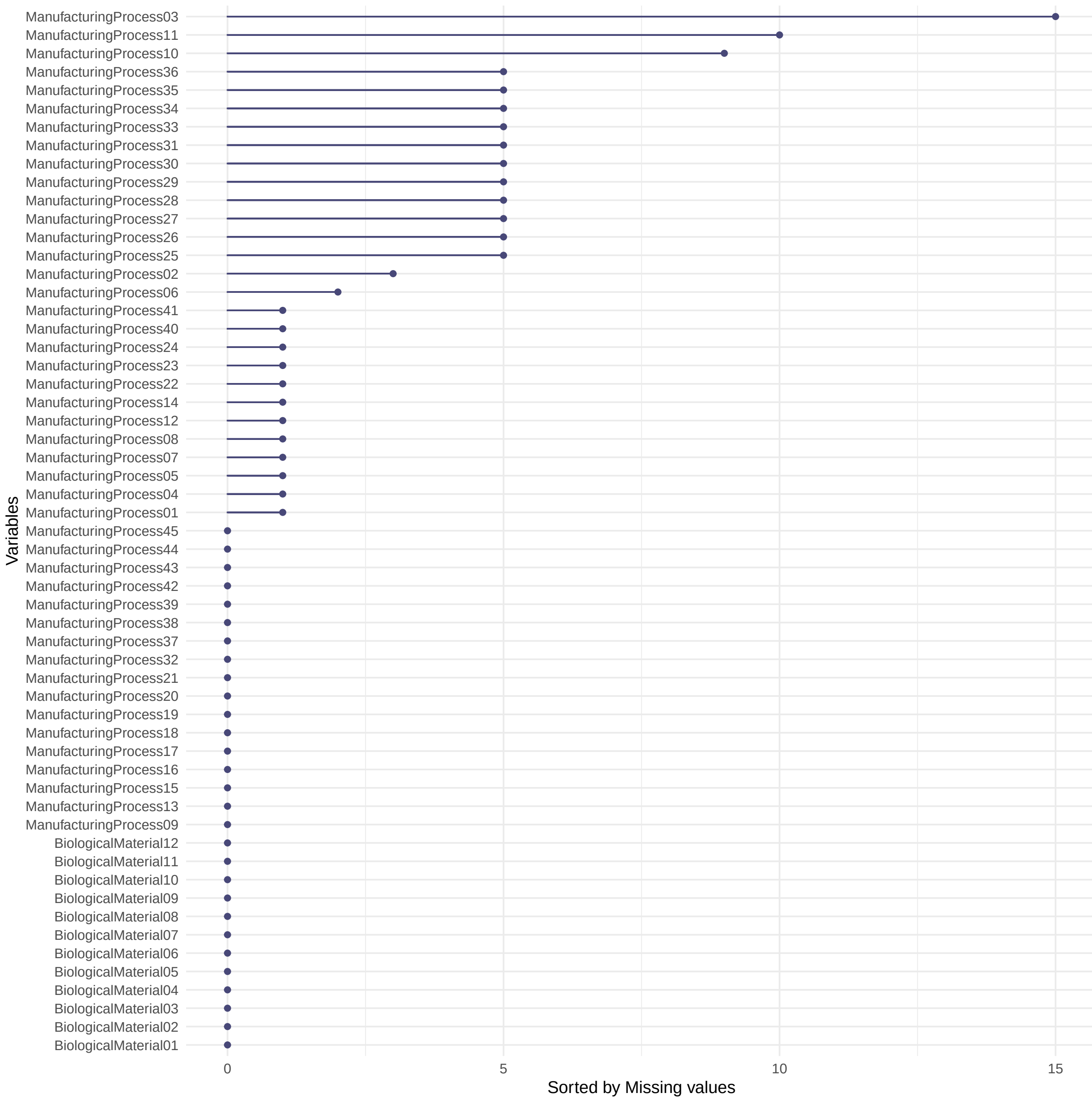
We will first see all the variables having any of the missing values. We have used below complete.cases() function to find the the missing values.

```
# columns having missing values
colnames(ChemicalManufacturingProcess)[!complete.cases(t(ChemicalManufacturingProcess))]
```

```
##  [1] "ManufacturingProcess01" "ManufacturingProcess02" "ManufacturingProcess03"
##  [4] "ManufacturingProcess04" "ManufacturingProcess05" "ManufacturingProcess06"
##  [7] "ManufacturingProcess07" "ManufacturingProcess08" "ManufacturingProcess10"
## [10] "ManufacturingProcess11" "ManufacturingProcess12" "ManufacturingProcess14"
## [13] "ManufacturingProcess22" "ManufacturingProcess23" "ManufacturingProcess24"
## [16] "ManufacturingProcess25" "ManufacturingProcess26" "ManufacturingProcess27"
## [19] "ManufacturingProcess28" "ManufacturingProcess29" "ManufacturingProcess30"
## [22] "ManufacturingProcess31" "ManufacturingProcess33" "ManufacturingProcess34"
## [25] "ManufacturingProcess35" "ManufacturingProcess36" "ManufacturingProcess40"
## [28] "ManufacturingProcess41"
```

So there are 28 columns having missing values. Here is the plot for missing values of all the predictors.

```
gg_miss_var(ChemicalManufacturingProcess[,-c(1)]) + labs(y = "Sorted by Missing values")
```



We will next use preProcess() method to impute the missing values using knnImpute (K nearest neighbor).

```
pre.proc <- preProcess(ChemicalManufacturingProcess[,c(-1)], method = "knnImpute")
chem_df <- predict(pre.proc, ChemicalManufacturingProcess[,c(-1)])
```

```
# columns having missing values
colnames(chem_df)[!complete.cases(t(chem_df))]
```

```
## character(0)
```

We will first filter out the predictors that have low frequencies using the `nearZeroVar` function from the caret package. After applying this function we see 1 column is removed and 56 predictors are left for modeling.

```
chem.remove.pred <- nearZeroVar(chem_df)
chem_df <- chem_df[,-chem.remove.pred]
length(chem.remove.pred) %>% paste('columns are removed. ', dim(chem_df)[2], ' predictors are left for m
```

```
## [1] "1 columns are removed.  56  predictors are left for modeling."
```

We will now look into pairwise correlation above 0.90 and remove the predictors having correlation with cutoff 0.90.

```
chem.corr.90 <- findCorrelation(cor(chem_df), cutoff=0.90)
chem_df <- chem_df[,-chem.corr.90]
length(chem.corr.90) %>% paste('columns having correlation 0.90 or more are removed. ', dim(chem_df)[2]
```

```
## [1] "10 columns having correlation 0.90 or more are removed.  46  predictors are left for modeling."
```

Next step is to split the data in training and testing set. We reserve 70% for training and 30% for testing. After split we will fit elastic net model.

```
set.seed(786)

pre.proc <- preProcess(chem_df, method = c("center", "scale"))
chem_df <- predict(pre.proc, chem_df)

# partition
chem.part <- createDataPartition(ChemicalManufacturingProcess$Yield, p=0.80, list = FALSE)

# predictor
X.train <- chem_df[chem.part,]
X.test <- chem_df[-chem.part,]

# response
y.train <- ChemicalManufacturingProcess$Yield[chem.part]
y.test <- ChemicalManufacturingProcess$Yield[-chem.part]
```

(a)

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

K-Nearest Neighbors

```
set.seed(317)
knnmodel <- train(X.train,
                y.train,
                method = "knn",
                preProcess = c("center","scale"),
                tuneLength = 10,
                trControl = trainControl(method = "cv"))

knnmodel
```

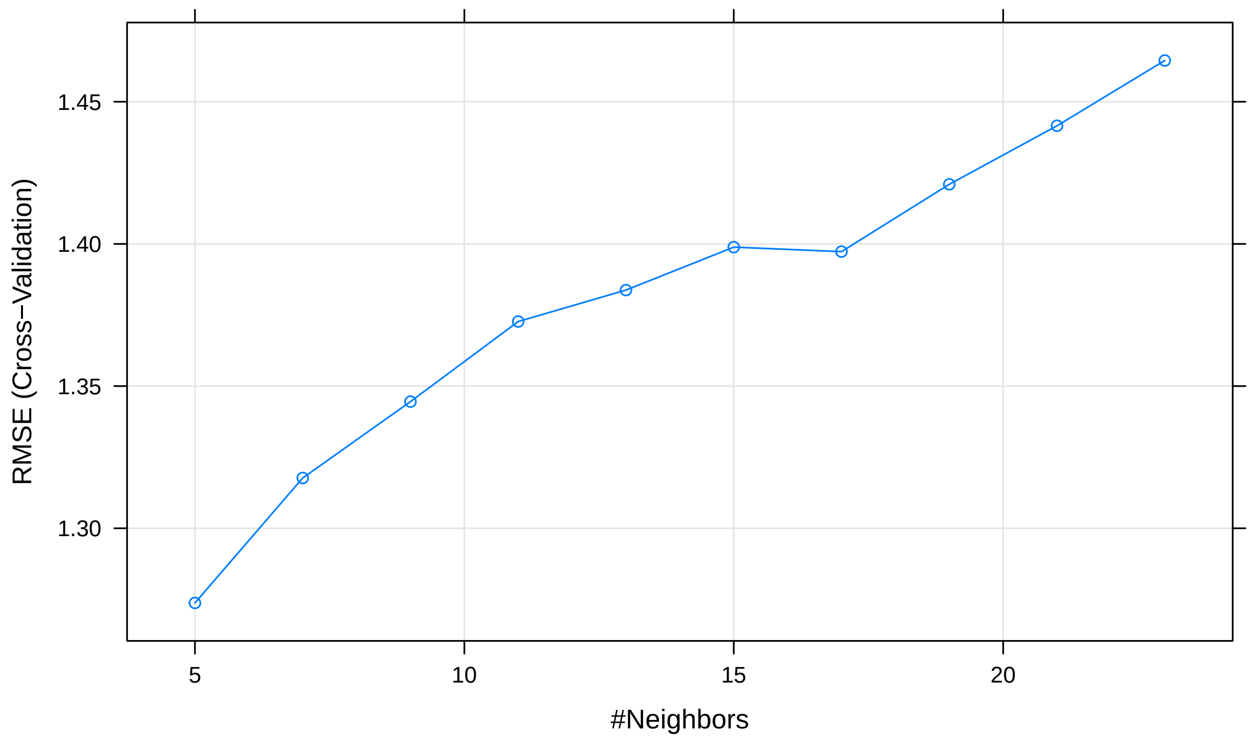
```
## k-Nearest Neighbors
##
## 144 samples
##  46 predictor
##
## Pre-processing: centered (46), scaled (46)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 129, 130, 130, 130, 130, ...
## Resampling results across tuning parameters:
##
```

```
##    k  RMSE      Rsquared   MAE
##    5  1.273751  0.5929874  1.028224
##    7  1.317688  0.5615865  1.075036
##    9  1.344539  0.5440593  1.100877
##   11  1.372702  0.5187307  1.130892
##   13  1.383772  0.5125153  1.120606
##   15  1.398863  0.4984152  1.132202
##   17  1.397293  0.5051418  1.141736
##   19  1.420958  0.4895823  1.154700
##   21  1.441547  0.4794607  1.180587
##   23  1.464504  0.4667482  1.198192
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.
```

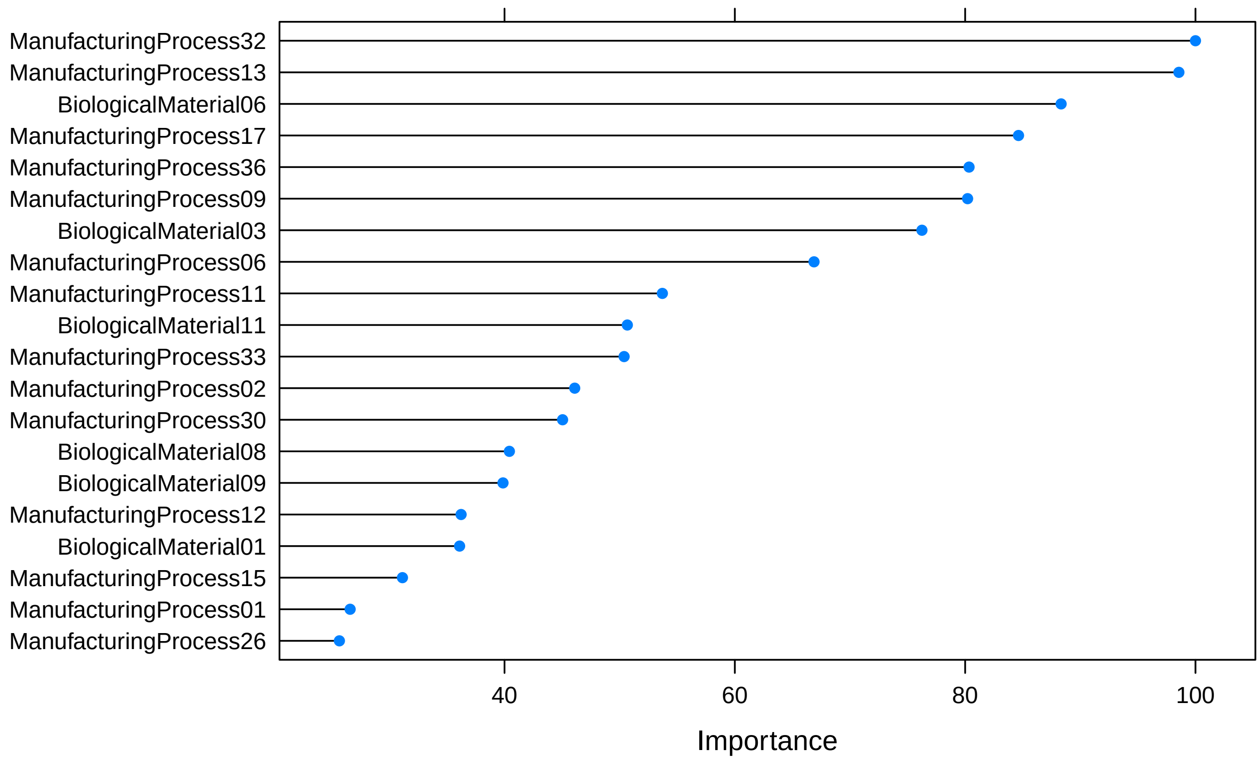
```r
# final parameters
knnmodel$bestTune
```

```
##   k
## 1 5
```

```r
# plot RMSE
plot(knnmodel)
```




```r
# plot variable importance
plot(varImp(knnmodel), top = 20)
```




```
data.frame(Rsquared=knnmodel[["results"]][["Rsquared"]][as.numeric(rownames(knnmodel$bestTune))],
           RMSE=knnmodel[["results"]][["RMSE"]][as.numeric(rownames(knnmodel$bestTune))])
```

```
##    Rsquared     RMSE
## 1 0.5929874 1.273751
```

The best tune parameter for the KNN model that resulted in the smallest root mean squared error is 5 which has RMSE $= 1.27$, and $R^2 = 0.52$. Also we can see quite a few top informative predictors from this model.

Support Vector Machines

```
set.seed(317)
svmmodel <- train(X.train,
                  y.train,
                  method = "svmRadial",
                  preProcess = c("center","scale"),
                  tuneLength = 10,
                  trControl = trainControl(method = "cv"))

svmmodel
```

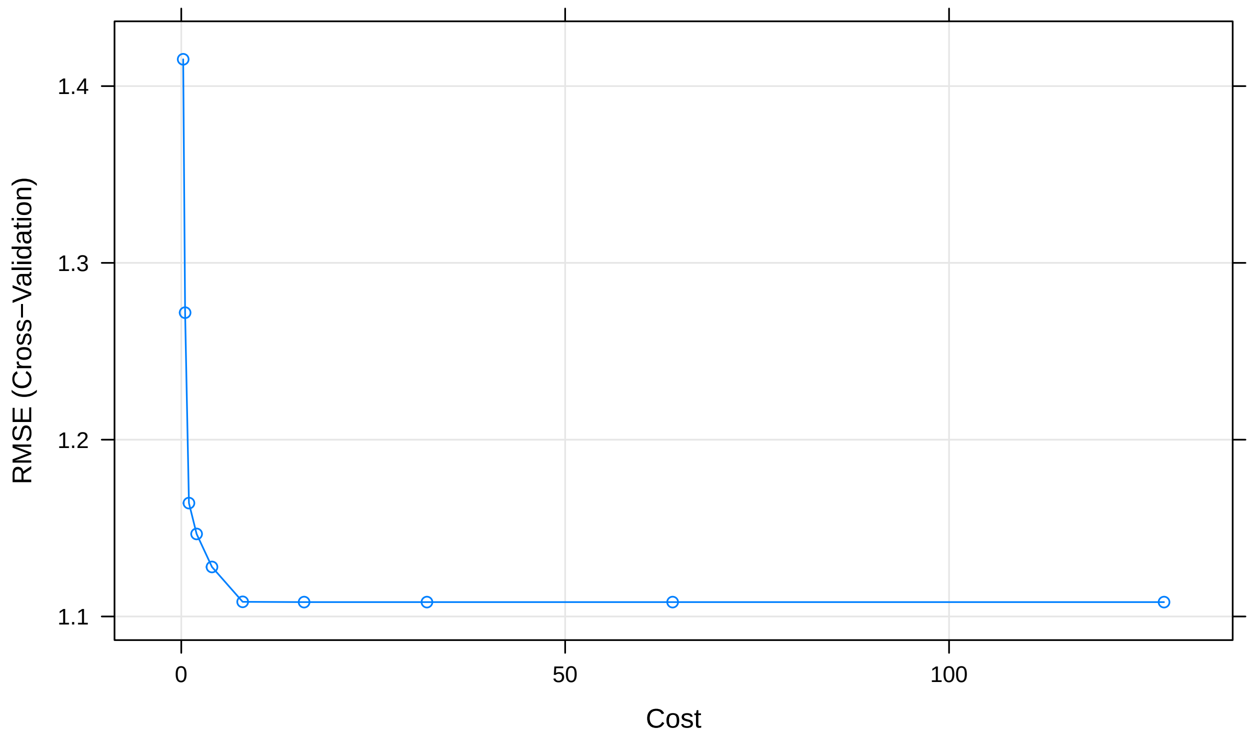
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 144 samples
##  46 predictor
##
## Pre-processing: centered (46), scaled (46)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 129, 130, 130, 130, 130, ...
## Resampling results across tuning parameters:
##
```

```
##   C       RMSE      Rsquared   MAE
##     0.25  1.415154  0.5598163  1.1451853
##     0.50  1.271848  0.6158726  1.0280135
##     1.00  1.164125  0.6704951  0.9387884
##     2.00  1.146672  0.6663610  0.9083151
##     4.00  1.128004  0.6700978  0.8930642
##     8.00  1.108284  0.6794459  0.8819400
##    16.00  1.108126  0.6796066  0.8818253
##    32.00  1.108126  0.6796066  0.8818253
##    64.00  1.108126  0.6796066  0.8818253
##   128.00  1.108126  0.6796066  0.8818253
##
## Tuning parameter 'sigma' was held constant at a value of 0.01657003
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.01657003 and C = 16.
```

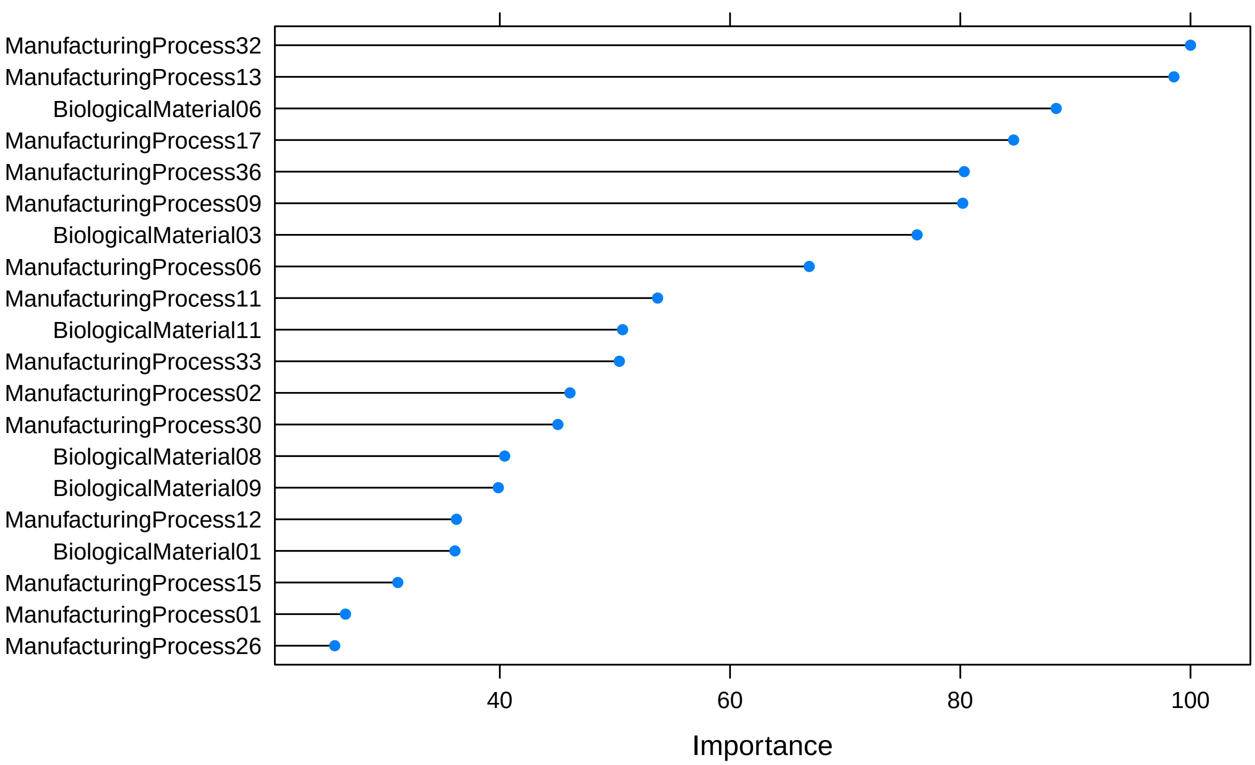
```
svmmodel$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 16
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0165700340906731
##
## Number of Support Vectors : 124
##
## Objective Function Value : -79.6507
## Training error : 0.009081
```

```
# plot RMSE
plot(svmmodel)
```




```r
# plot variable importance
plot(varImp(svmmodel), top = 20)
```




```r
data.frame(Rsquared=svmmodel[["results"]][["Rsquared"]][as.numeric(rownames(svmmodel$bestTune))],
           RMSE=svmmodel[["results"]][["RMSE"]][as.numeric(rownames(svmmodel$bestTune))])
```

```
##    Rsquared     RMSE
## 1 0.6796066 1.108126
```

So we can see here that best SVM model produced Rsquared as 0.68 and RMSE as 1.11. Tuning parameter 'sigma' was held constant at a value of 0.016 RMSE was used to select the optimal model using the smallest value. The final values used for the model were sigma = 0.01657003 and C = 16.

Multivariate Adaptive Regression Splines

```r
set.seed(317)
marsGrid2 <- expand.grid(.degree=1:2, .nprune=2:38)
marsmodel <- train(X.train,
                y.train,
                method = "earth",
                #preProcess = c("center","scale"),
                tuneGrid = marsGrid2,
                trControl = trainControl(method = "cv"))

marsmodel
```

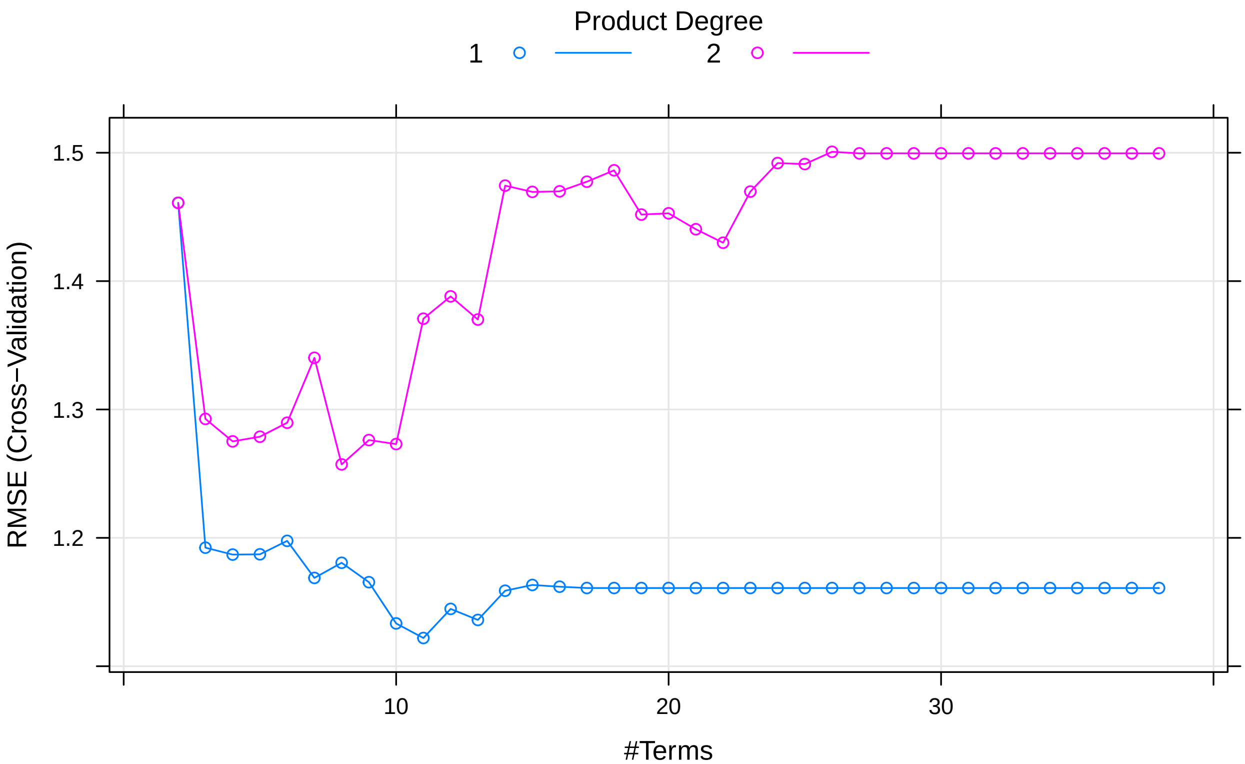
```
## Multivariate Adaptive Regression Spline
##
## 144 samples
##  46 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 129, 130, 130, 130, 130, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      1.460974  0.4215160  1.1637009
##   1        3      1.192352  0.6170315  0.9521976
##   1        4      1.186962  0.6105407  0.9393440
##   1        5      1.187175  0.6151844  0.9541531
##   1        6      1.197656  0.5981361  0.9579706
##   1        7      1.168795  0.6288465  0.9498812
##   1        8      1.180580  0.6346928  0.9612948
##   1        9      1.165459  0.6392418  0.9483000
##   1       10      1.133341  0.6569746  0.9259037
##   1       11      1.121970  0.6619767  0.8947102
##   1       12      1.144647  0.6538257  0.9034170
##   1       13      1.136059  0.6721578  0.8940545
##   1       14      1.158815  0.6559312  0.9097781
##   1       15      1.163309  0.6509330  0.9124696
##   1       16      1.161948  0.6520535  0.9114202
##   1       17      1.160917  0.6524624  0.9082088
##   1       18      1.160917  0.6524624  0.9082088
##   1       19      1.160917  0.6524624  0.9082088
##   1       20      1.160917  0.6524624  0.9082088
##   1       21      1.160917  0.6524624  0.9082088
##   1       22      1.160917  0.6524624  0.9082088
##   1       23      1.160917  0.6524624  0.9082088
##   1       24      1.160917  0.6524624  0.9082088
##   1       25      1.160917  0.6524624  0.9082088
```

```
##   1        26      1.160917  0.6524624  0.9082088
##   1        27      1.160917  0.6524624  0.9082088
##   1        28      1.160917  0.6524624  0.9082088
##   1        29      1.160917  0.6524624  0.9082088
##   1        30      1.160917  0.6524624  0.9082088
##   1        31      1.160917  0.6524624  0.9082088
##   1        32      1.160917  0.6524624  0.9082088
##   1        33      1.160917  0.6524624  0.9082088
##   1        34      1.160917  0.6524624  0.9082088
##   1        35      1.160917  0.6524624  0.9082088
##   1        36      1.160917  0.6524624  0.9082088
##   1        37      1.160917  0.6524624  0.9082088
##   1        38      1.160917  0.6524624  0.9082088
##   2         2      1.460974  0.4215160  1.1637009
##   2         3      1.292673  0.5439056  1.0242094
##   2         4      1.275148  0.5800433  1.0139713
##   2         5      1.278785  0.5949042  1.0109620
##   2         6      1.289694  0.5872296  1.0161207
##   2         7      1.340280  0.5519717  1.0505082
##   2         8      1.257209  0.6027248  1.0081118
##   2         9      1.276170  0.5690879  1.0233126
##   2        10      1.273060  0.5844886  1.0159144
##   2        11      1.370720  0.5606387  1.0456423
##   2        12      1.388115  0.5557170  1.0549758
##   2        13      1.370067  0.5623987  1.0473908
##   2        14      1.474397  0.5387508  1.0732628
##   2        15      1.469498  0.5442746  1.0848046
##   2        16      1.469930  0.5571116  1.0793236
##   2        17      1.477451  0.5477182  1.0879546
##   2        18      1.486312  0.5443474  1.0963863
##   2        19      1.451833  0.5618974  1.0693779
##   2        20      1.452800  0.5626639  1.0768835
##   2        21      1.440408  0.5683565  1.0651024
##   2        22      1.429902  0.5793222  1.0456420
##   2        23      1.469748  0.5636573  1.0642625
##   2        24      1.491947  0.5537047  1.0831403
##   2        25      1.491180  0.5539623  1.0839457
##   2        26      1.500749  0.5523431  1.0750270
##   2        27      1.499503  0.5512516  1.0759555
##   2        28      1.499503  0.5512516  1.0759555
##   2        29      1.499503  0.5512516  1.0759555
##   2        30      1.499503  0.5512516  1.0759555
##   2        31      1.499503  0.5512516  1.0759555
##   2        32      1.499503  0.5512516  1.0759555
##   2        33      1.499503  0.5512516  1.0759555
##   2        34      1.499503  0.5512516  1.0759555
##   2        35      1.499503  0.5512516  1.0759555
##   2        36      1.499503  0.5512516  1.0759555
##   2        37      1.499503  0.5512516  1.0759555
##   2        38      1.499503  0.5512516  1.0759555
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 11 and degree = 1.
```

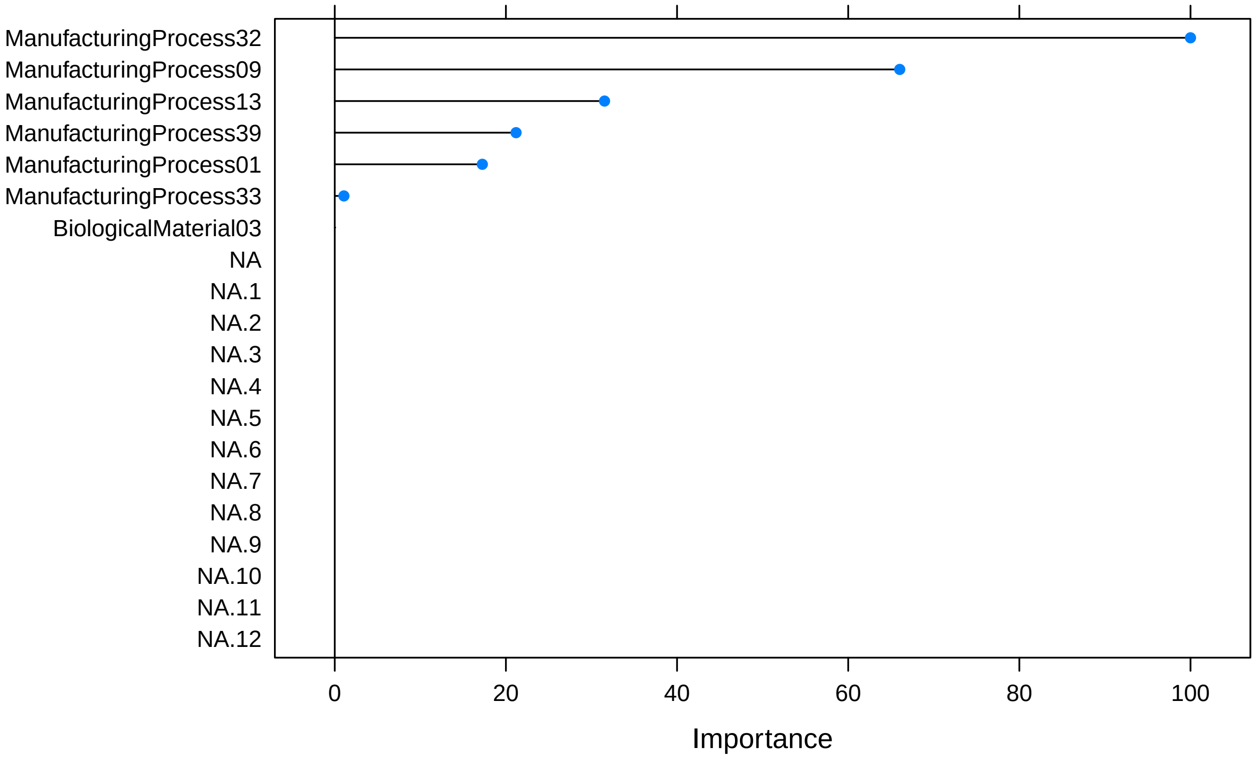
```
# final parameters
marsmodel$bestTune
```

```
##    nprune degree
## 10     11      1
```

```
plot(marsmodel)
```




```
# plot variable importance
plot(varImp(marsmodel), top=20)
```




```
data.frame(Rsquared=marsmodel[["results"]][["Rsquared"]][as.numeric(rownames(marsmodel$bestTune))],
           RMSE=marsmodel[["results"]][["RMSE"]][as.numeric(rownames(marsmodel$bestTune))])
```

```
##    Rsquared     RMSE
## 1 0.5872296 1.289694
```

RMSE was used to select the optimal model using the smallest value. The final values used for the model were nprune = 11 and degree = 1 that resulted Rsquared as 0.59 and RMSE as 1.29. So far we can see SVM model has a best fit on training data comparing with KNN and MARS. Also we see 4 top predictors in this model.

Neural Networks

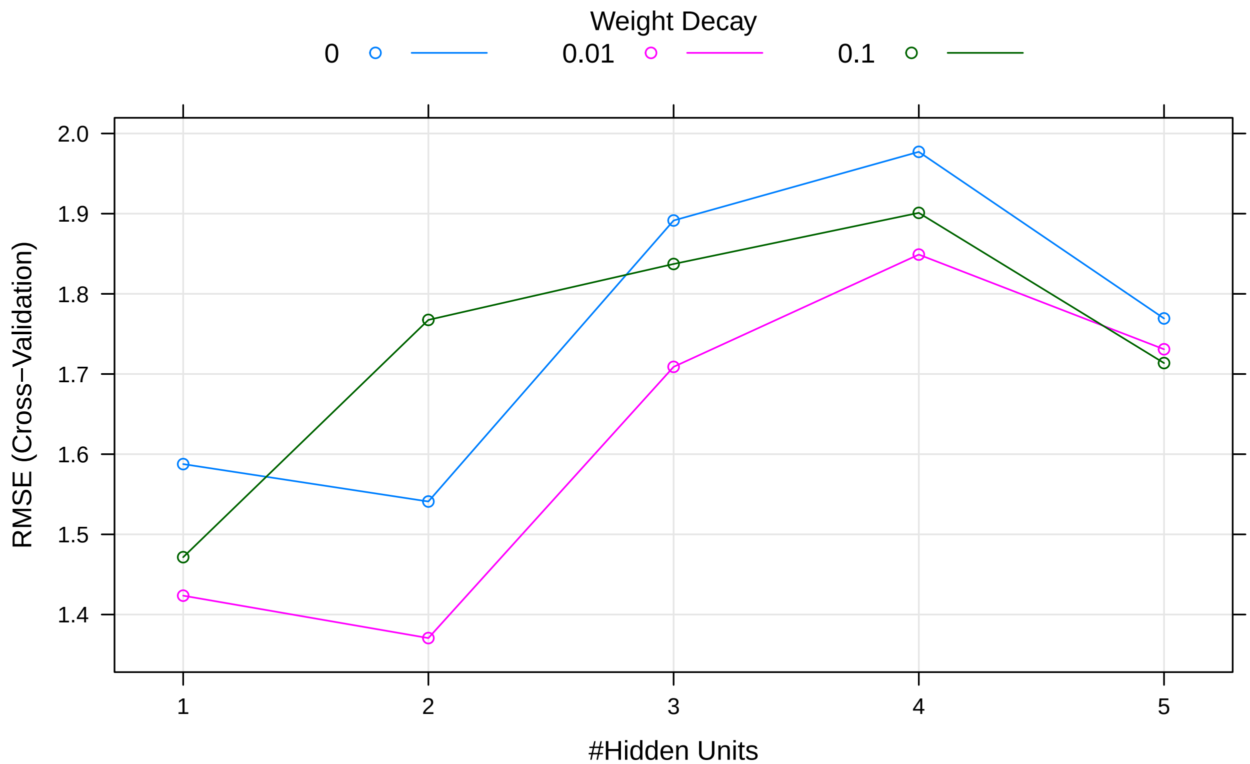
```
set.seed(317)
nnetGrid2 <- expand.grid(.decay=c(0,0.01,.1),
                         .size=c(1:5),
                         .bag=FALSE)

nnetmodel <- train(X.train,
                   y.train,
                   method = "avNNet",
                   tuneGrid = nnetGrid2,
                   #preProcess = c("center","scale"),
                   trControl = trainControl(method = "cv"),
                   linout = TRUE,
                   trace = FALSE,
                   MaxNWts =5 * (ncol(X.train)+1) +5+1,
                   maxit=500)

nnetmodel
```

```
## Model Averaged Neural Network
##
## 144 samples
##  46 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 129, 130, 130, 130, 130, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE      Rsquared   MAE
##   0.00   1     1.587654  0.3952524  1.311361
##   0.00   2     1.540967  0.3746445  1.251953
##   0.00   3     1.891477  0.3564232  1.504976
##   0.00   4     1.977102  0.3060809  1.548088
##   0.00   5     1.769296  0.4506122  1.352078
##   0.01   1     1.423551  0.5161222  1.158160
##   0.01   2     1.370648  0.5601359  1.128443
##   0.01   3     1.708938  0.4570819  1.287134
##   0.01   4     1.848971  0.4322456  1.381563
##   0.01   5     1.730875  0.4363947  1.334462
##   0.10   1     1.471556  0.5239966  1.133428
##   0.10   2     1.767478  0.4974421  1.295874
##   0.10   3     1.837230  0.4563215  1.299786
##   0.10   4     1.900986  0.4101949  1.403459
##   0.10   5     1.713670  0.4407929  1.262127
##
## 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 = 2, decay = 0.01 and bag = FALSE.
```

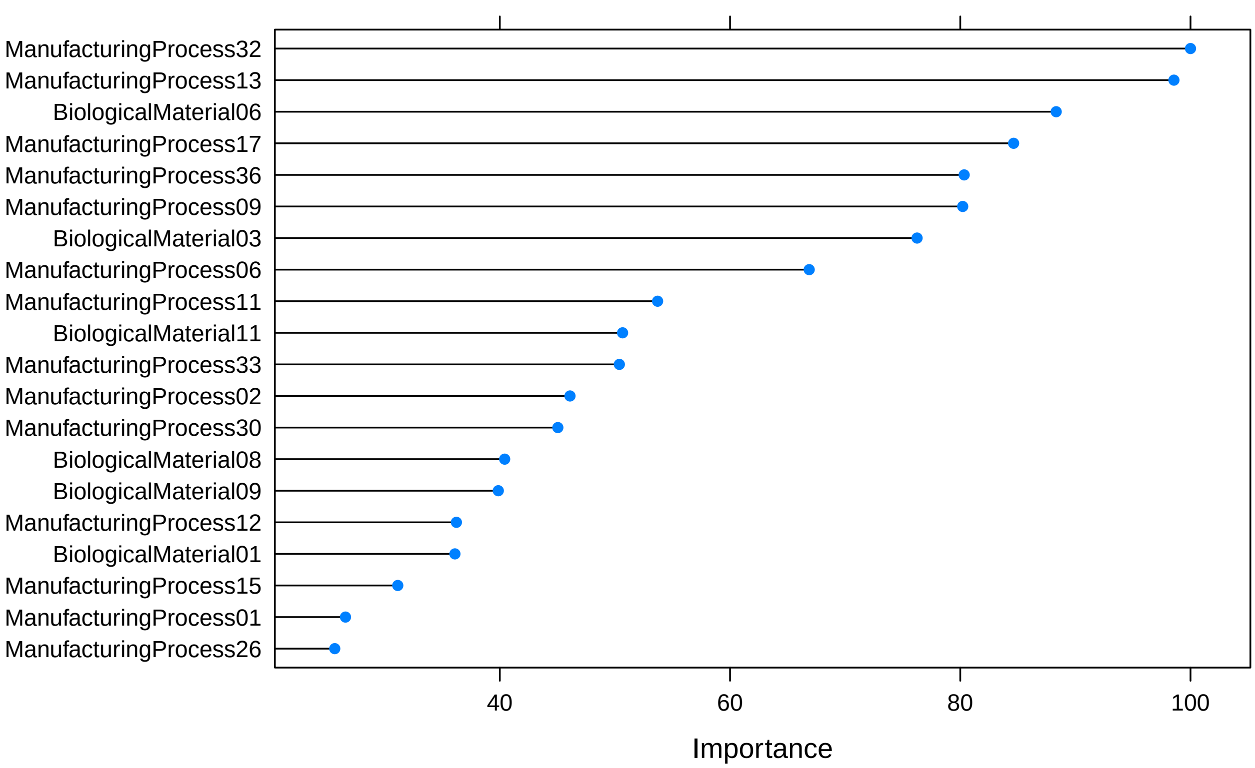
```r
# final parameters
nnetmodel$bestTune
```

```
##   size decay   bag
## 7    2  0.01 FALSE
```

```r
# plot RMSE
plot(nnetmodel)
```




```r
# plot variable importance
plot(varImp(nnetmodel), top=20)
```




```r
data.frame(Rsquared=nnetmodel[["results"]][["Rsquared"]][as.numeric(rownames(nnetmodel$bestTune))],
```

```
          RMSE=nnetmodel[["results"]][["RMSE"]][as.numeric(rownames(nnetmodel$bestTune))])
```

```
##    Rsquared     RMSE
## 1 0.3564232 1.891477
```

RMSE was used to select the optimal model using the smallest value. Tuning parameter 'bag' was held constant at a value of FALSE. The final values used for the model were size = 2, decay = 0.01 and bag = FALSE that resulted the Rsquared 0.36 and RMSE as 1.89.

Optimal resampling

Now we will use resampling method to get the performance metrics and analyze the results to select the best fit model here. So far SVM model produced the best results.

```
set.seed(317)
summary(resamples(list(KNN=knnmodel, SVM=svmmodel, MARS=marsmodel, NNET=nnetmodel)))
```

```
##
## Call:
## summary.resamples(object = resamples(list(KNN = knnmodel, SVM = svmmodel,
##  MARS = marsmodel, NNET = nnetmodel)))
##
## Models: KNN, SVM, MARS, NNET
## Number of resamples: 10
##
## MAE
##           Min.   1st Qu.    Median      Mean  3rd Qu.     Max. NA's
## KNN  0.7812889 0.9194152 1.0265000 1.0282238 1.173248 1.255548    0
## SVM  0.6364848 0.7244008 0.8391784 0.8818253 1.014569 1.234649    0
## MARS 0.4675961 0.7462232 0.9176210 0.8947102 1.027591 1.189657    0
## NNET 0.8488526 1.0329065 1.1097302 1.1284429 1.264049 1.361221    0
##
## RMSE
##           Min.   1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## KNN  0.9090237 1.1409380 1.293382 1.273751 1.473110 1.565858    0
## SVM  0.8671916 0.9272013 1.041208 1.108126 1.236918 1.601517    0
## MARS 0.6428867 0.9272217 1.123618 1.121970 1.242323 1.556874    0
## NNET 1.0669936 1.2703743 1.367912 1.370648 1.474480 1.674919    0
##
## Rsquared
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## KNN  0.4322442 0.5023306 0.5819276 0.5929874 0.6600140 0.8488601    0
## SVM  0.5176962 0.6010184 0.6806670 0.6796066 0.7626151 0.8122909    0
## MARS 0.4147210 0.5503631 0.6823146 0.6619767 0.7852096 0.8608205    0
## NNET 0.3594460 0.4686967 0.6189848 0.5601359 0.6358781 0.6723116    0
```

Test set performance

```
set.seed(317)
knnpred <- predict(knnmodel, newdata = X.test)
svmpred <- predict(svmmodel, newdata = X.test)
marspred <- predict(marsmodel, newdata = X.test)
nnetpred <- predict(nnetmodel, newdata = X.test)
```

```r
data.frame(rbind(KNN=postResample(pred=knnpred,obs = y.test),
                 SVM=postResample(pred=svmpred,obs = y.test),
                 MARS=postResample(pred=marspred,obs = y.test),
                 NNET=postResample(pred=nnetpred,obs = y.test)))
```

```
##          RMSE  Rsquared       MAE
## KNN  1.071578 0.5838633 0.8358750
## SVM  1.027715 0.6135779 0.8677946
## MARS 1.079173 0.5890563 0.8368743
## NNET 1.087833 0.6233270 0.9156897
```

From the results, we can conclude that the SVM model predicted the test response with best accuracy R^2=0.62, RMSE=1.02 and MAE=0.86

(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?

Here is the list of top 10 most important predictors from SVM model. The `caret:varImp` calculates the variable importance for regression that shows the relationship between each predictor and the output from linear model fit. We can see below the most important contribution variable is `ManufacturingProcess32` and hence ManufacturingProcess dominate the list.

```r
# plot variable importance
varImp(svmmodel, top=10)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 46)
##
##                        Overall
## ManufacturingProcess32  100.00
## ManufacturingProcess13   98.56
## BiologicalMaterial06     88.33
## ManufacturingProcess17   84.64
## ManufacturingProcess36   80.34
## ManufacturingProcess09   80.21
## BiologicalMaterial03     76.25
## ManufacturingProcess06   66.88
## ManufacturingProcess11   53.71
## BiologicalMaterial11     50.67
## ManufacturingProcess33   50.38
## ManufacturingProcess02   46.10
## ManufacturingProcess30   45.04
## BiologicalMaterial08     40.42
## BiologicalMaterial09     39.87
## ManufacturingProcess12   36.23
## BiologicalMaterial01     36.10
## ManufacturingProcess15   31.14
## ManufacturingProcess01   26.59
## ManufacturingProcess26   25.66
```

It was stated earlier that elasticnwt model that best fitted the data among linear models. We can see here too that ManufacturingProcess variables dominates the list but ranks seem different between linear and non linear models.

```
# tune elastic net model
chem.enet.fit <- train(x=X.train,
                       y=y.train,
                       method="glmnet",
                       metric="Rsquared",
                       trControl=trainControl(method = "cv",number=10),
                       tuneLength = 5
                 )


varImp(chem.enet.fit)
```

```
## glmnet variable importance
##
##   only 20 most important variables shown (out of 46)
##
##                          Overall
## ManufacturingProcess32 100.00000
## ManufacturingProcess09  56.42573
## ManufacturingProcess13  35.05073
## ManufacturingProcess36  24.00339
## ManufacturingProcess17  22.47464
## BiologicalMaterial06    18.92913
## ManufacturingProcess06   2.94998
## ManufacturingProcess39   0.64278
## ManufacturingProcess44   0.04296
## ManufacturingProcess01   0.00000
## ManufacturingProcess24   0.00000
## ManufacturingProcess11   0.00000
## ManufacturingProcess16   0.00000
## ManufacturingProcess22   0.00000
## ManufacturingProcess21   0.00000
## BiologicalMaterial01     0.00000
## ManufacturingProcess30   0.00000
## ManufacturingProcess38   0.00000
## ManufacturingProcess34   0.00000
## ManufacturingProcess10   0.00000
```

(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?

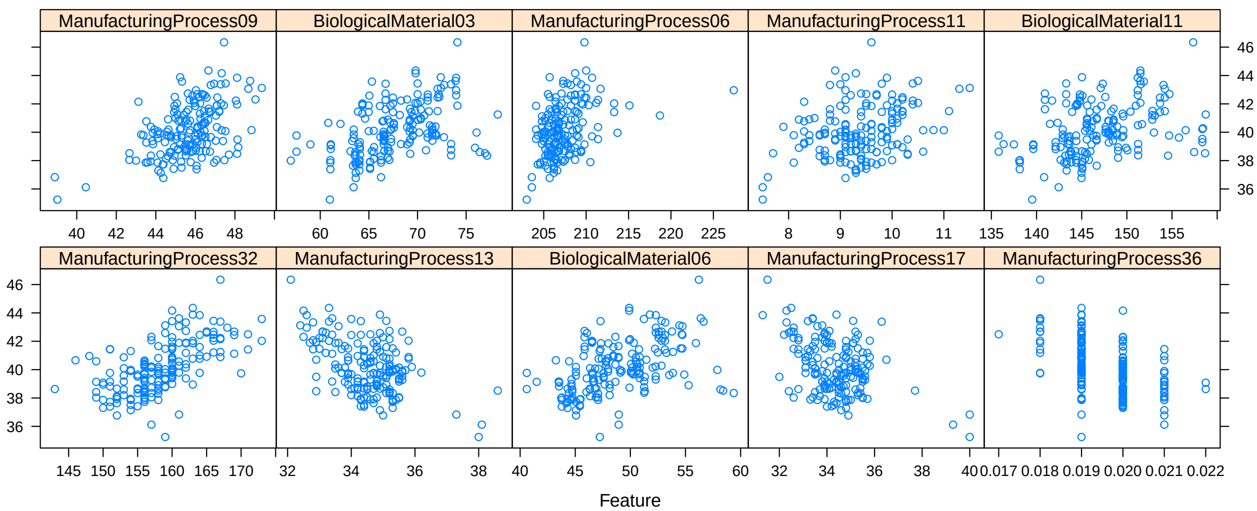
We will now get the top 10 predictors, arrange it in order and then draw the `featureplot` to explore the visualization.

```
# predictors importance
vimp <- varImp(svmmodel)$importance
# top 10 predictors
top10.vars <- head(rownames(vimp)[order(-vimp$Overall)], 10)
as.data.frame(top10.vars)
```

```
##                 top10.vars
```

```
## 1  ManufacturingProcess32
## 2  ManufacturingProcess13
## 3    BiologicalMaterial06
## 4  ManufacturingProcess17
## 5  ManufacturingProcess36
## 6  ManufacturingProcess09
## 7    BiologicalMaterial03
## 8  ManufacturingProcess06
## 9  ManufacturingProcess11
## 10   BiologicalMaterial11
```

```
X <- ChemicalManufacturingProcess[,top10.vars]
Y <- ChemicalManufacturingProcess$Yield

featurePlot(X,Y)
```



From the plots above, it is apparent that for SVM model (optimal model) the top predictors have mostly linear relationship with the response `Yield`. Increasing the features like `ManufacturingProcess32` or `BiologicalMaterial06` increases the response while increasing features like ManufacturingProcess13 cause decrease in response variable.