exercise72

Lev Mazaev

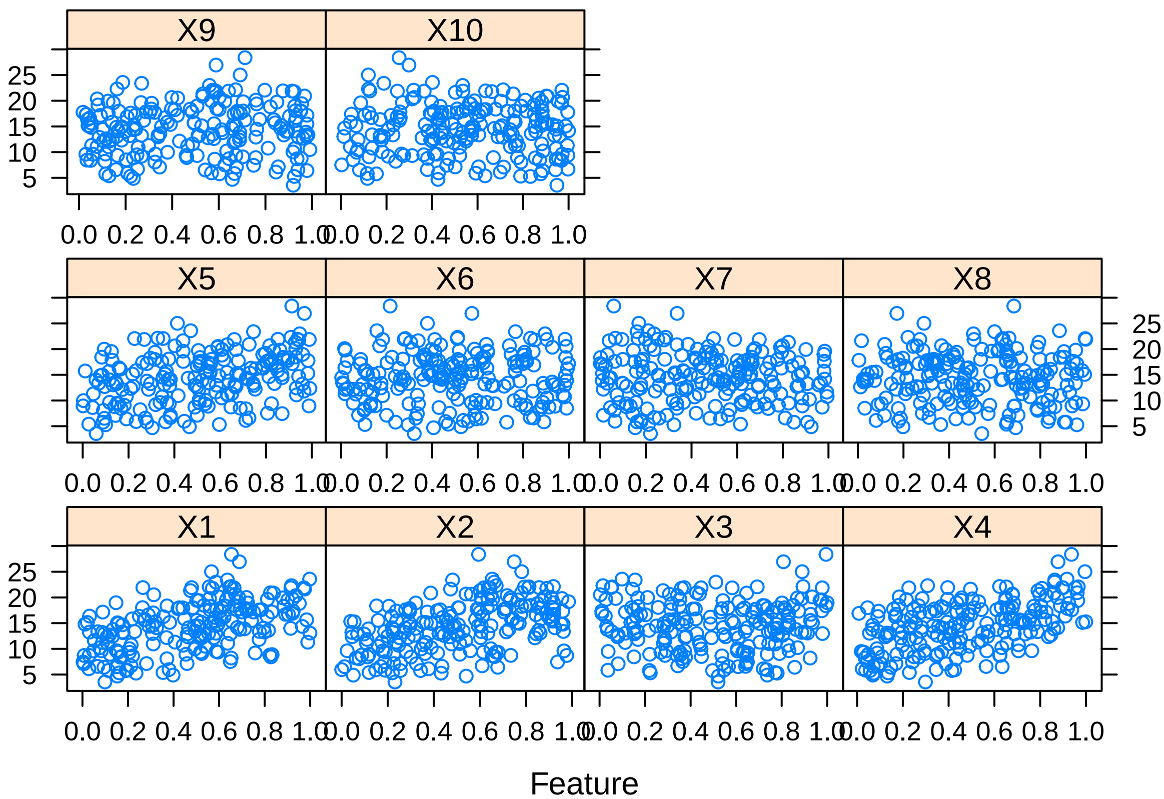
Exercise 7.2 (Applied Predictive Modeling, p. 169)

Loading the packages

```
library(mlbench)
library(caret)
library(doMC)
library(earth)
registerDoMC(8)
```

Creating the data

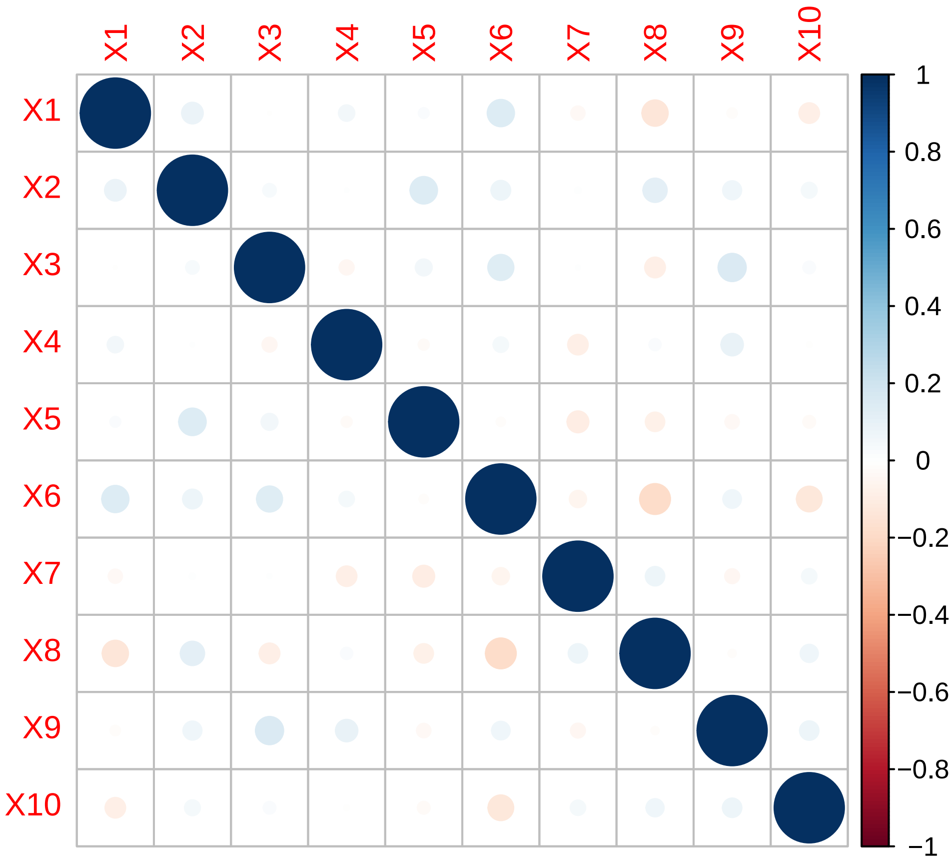
```
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)
trainingData$x <- data.frame(trainingData$x)
featurePlot(trainingData$x, trainingData$y)
```




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

Correlation check

```
corrplot::corrplot(cor(trainingData$x))
```




```
findCorrelation(cor(trainingData$x), cutoff = 0.75)
```

```
## integer(0)
```

There are no highly-correlated predictors

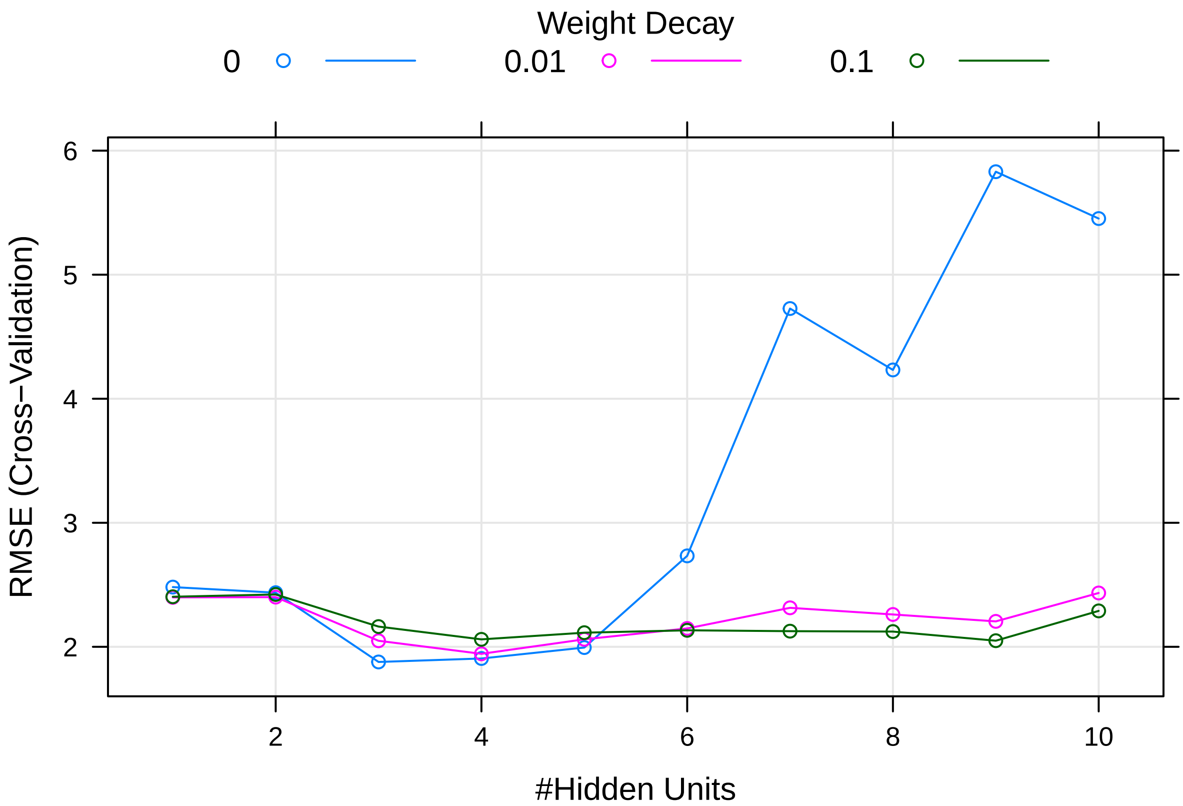
Neural Network

```
nnetGrid <- expand.grid(.decay = c(0, 0.01, .1), .size = 1:10, .bag = FALSE)
nnetModel <- train(x = trainingData$x, y = trainingData$y,
                   method = "avNNet", tuneGrid = nnetGrid,
                   trControl = trainControl(method = "cv"),
                   preProcess = c("center", "scale"),
                   linout = TRUE, trace = TRUE,
                   MaxNWts = 10 * (ncol(trainingData$x) + 1) + 10 + 1,
                   maxit = 500)
nnetModel
```

```
## Model Averaged Neural Network
##
## 200 samples
```

```
##  10 predictors
##
## 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:
##
##   decay  size  RMSE      Rsquared   MAE
##   0.00    1    2.481454  0.7469747  1.975542
##   0.00    2    2.436424  0.7649192  1.930859
##   0.00    3    1.877814  0.8573051  1.476834
##   0.00    4    1.905918  0.8541738  1.529084
##   0.00    5    1.993979  0.8409575  1.576757
##   0.00    6    2.733189  0.7264896  2.029163
##   0.00    7    4.727163  0.5186760  2.950852
##   0.00    8    4.232154  0.6227455  2.886581
##   0.00    9    5.830266  0.4324491  3.391461
##   0.00   10    5.452456  0.5700175  2.969528
##   0.01    1    2.398640  0.7703358  1.871041
##   0.01    2    2.400588  0.7710578  1.888898
##   0.01    3    2.048665  0.8326338  1.617814
##   0.01    4    1.943547  0.8453374  1.564313
##   0.01    5    2.060612  0.8232218  1.568054
##   0.01    6    2.147995  0.8115418  1.721255
##   0.01    7    2.314310  0.7908798  1.836394
##   0.01    8    2.260879  0.7998637  1.815767
##   0.01    9    2.205192  0.8016452  1.794733
##   0.01   10    2.433887  0.7698170  1.895905
##   0.10    1    2.403712  0.7687049  1.869539
##   0.10    2    2.421932  0.7689830  1.888883
##   0.10    3    2.163172  0.8143900  1.700189
##   0.10    4    2.060149  0.8297096  1.670812
##   0.10    5    2.113895  0.8213375  1.700141
##   0.10    6    2.133011  0.8251511  1.710712
##   0.10    7    2.126111  0.8217688  1.685677
##   0.10    8    2.122994  0.8237300  1.706942
##   0.10    9    2.049278  0.8280368  1.611528
##   0.10   10    2.289593  0.7878086  1.811428
##
## 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 = 3, decay = 0 and bag
##  = FALSE.
```

```r
plot(nnetModel)
```



Performance evaluation

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

```
##      RMSE  Rsquared       MAE
## 1.9263163 0.8528292 1.4630365
```

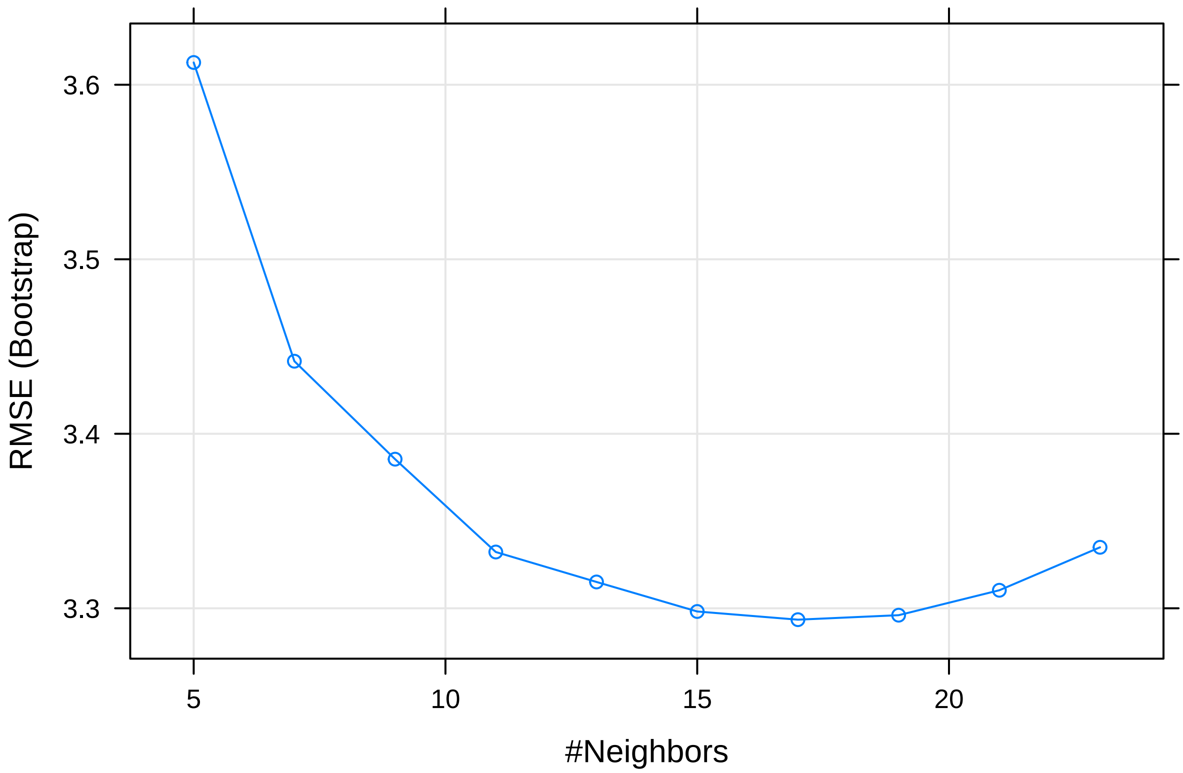
K Nearest Neighbors

```
knnModel <- train(x = trainingData$x, y = trainingData$y,
                  method = "knn", preProcess = c("center", "scale"),
                  tuneLength = 10)
knnModel
```

```
## k-Nearest Neighbors
##
## 200 samples
##  10 predictors
##
## 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  3.612753  0.4847321  2.934855
##     7  3.441582  0.5351824  2.792389
##     9  3.385434  0.5592947  2.740026
##    11  3.332242  0.5864511  2.700934
##    13  3.315092  0.6029810  2.686746
##    15  3.298171  0.6224845  2.674603
##    17  3.293481  0.6341255  2.664912
##    19  3.296045  0.6483953  2.676133
##    21  3.310329  0.6539922  2.694153
##    23  3.334949  0.6560518  2.714964
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 17.
```

```r
plot(knnModel)
```



Performance evaluation

```r
knnPred <- predict(knnModel, newdata = testData$x)
modelstats <- rbind(modelstats, postResample(pred = knnPred, obs = testData$y))
postResample(pred = knnPred, obs = testData$y)
```

```
##      RMSE  Rsquared       MAE
## 3.2040595 0.6819919 2.5683461
```

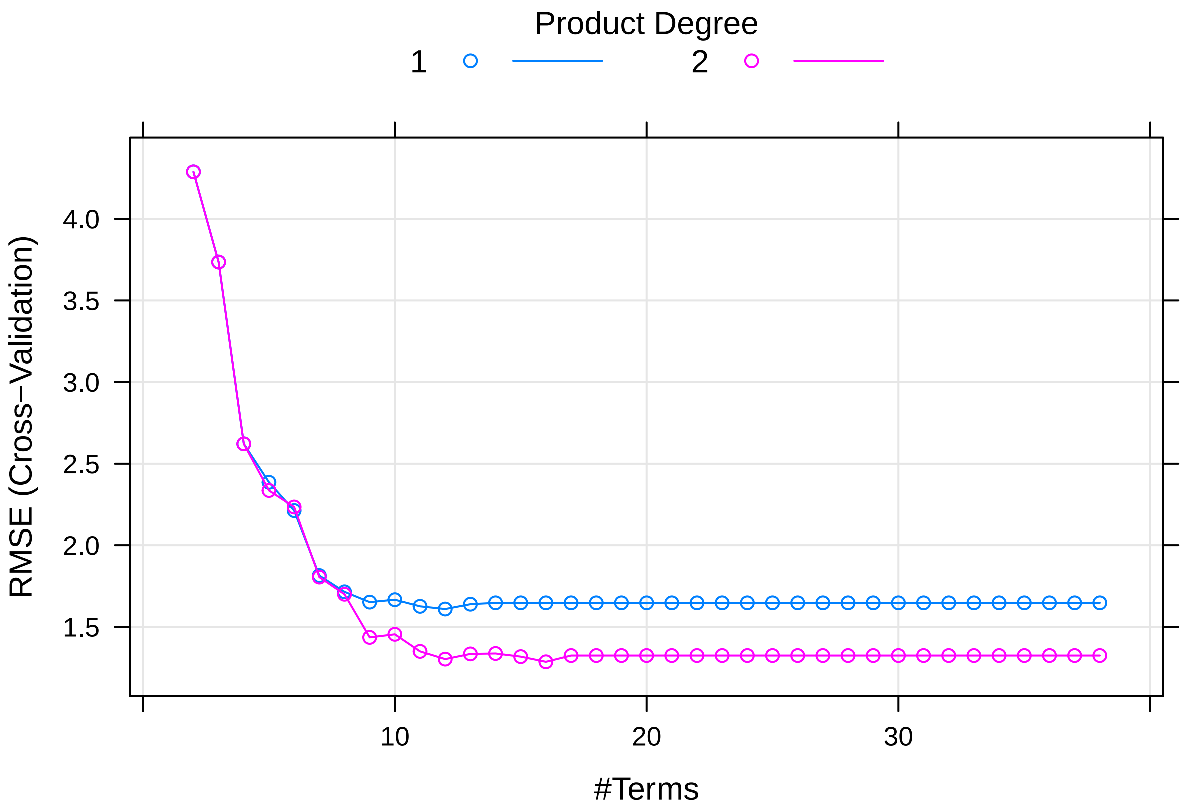
MARS

```r
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)
marsModel <- train(x = trainingData$x, y = trainingData$y, method = "earth",
                   tuneGrid = marsGrid, trControl = trainControl(method = "cv"))
marsModel
```

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictors
##
## No pre-processing
## 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      4.287738  0.2766228  3.541096
##   1        3      3.735295  0.4579537  3.037607
##   1        4      2.621465  0.7348671  2.115178
##   1        5      2.386064  0.7837287  1.928445
##   1        6      2.213116  0.8247552  1.769175
##   1        7      1.814470  0.8776482  1.436171
##   1        8      1.715089  0.8867568  1.344378
##   1        9      1.651983  0.8946079  1.296225
##   1       10      1.666730  0.8896019  1.324865
##   1       11      1.626214  0.8938909  1.285840
##   1       12      1.609580  0.8962215  1.284743
##   1       13      1.639251  0.8929914  1.297566
##   1       14      1.647570  0.8916943  1.301484
##   1       15      1.647570  0.8916943  1.301484
##   1       16      1.647570  0.8916943  1.301484
##   1       17      1.647570  0.8916943  1.301484
##   1       18      1.647570  0.8916943  1.301484
##   1       19      1.647570  0.8916943  1.301484
##   1       20      1.647570  0.8916943  1.301484
##   1       21      1.647570  0.8916943  1.301484
##   1       22      1.647570  0.8916943  1.301484
##   1       23      1.647570  0.8916943  1.301484
##   1       24      1.647570  0.8916943  1.301484
##   1       25      1.647570  0.8916943  1.301484
##   1       26      1.647570  0.8916943  1.301484
##   1       27      1.647570  0.8916943  1.301484
##   1       28      1.647570  0.8916943  1.301484
##   1       29      1.647570  0.8916943  1.301484
##   1       30      1.647570  0.8916943  1.301484
##   1       31      1.647570  0.8916943  1.301484
##   1       32      1.647570  0.8916943  1.301484
##   1       33      1.647570  0.8916943  1.301484
##   1       34      1.647570  0.8916943  1.301484
##   1       35      1.647570  0.8916943  1.301484
##   1       36      1.647570  0.8916943  1.301484
##   1       37      1.647570  0.8916943  1.301484
##   1       38      1.647570  0.8916943  1.301484
##   2        2      4.287738  0.2766228  3.541096
```

```
##   2         3      3.735295  0.4579537  3.037607
##   2         4      2.621465  0.7348671  2.115178
##   2         5      2.336068  0.7905828  1.886709
##   2         6      2.235016  0.8153197  1.740073
##   2         7      1.805188  0.8775674  1.440077
##   2         8      1.700080  0.8929903  1.314594
##   2         9      1.436055  0.9190680  1.136430
##   2        10      1.454841  0.9172529  1.143233
##   2        11      1.350271  0.9318665  1.047642
##   2        12      1.302893  0.9353308  1.032536
##   2        13      1.334764  0.9336181  1.040403
##   2        14      1.337295  0.9323946  1.049709
##   2        15      1.318129  0.9361413  1.047690
##   2        16      1.286297  0.9375534  1.016883
##   2        17      1.324754  0.9350295  1.038155
##   2        18      1.324754  0.9350295  1.038155
##   2        19      1.324754  0.9350295  1.038155
##   2        20      1.324754  0.9350295  1.038155
##   2        21      1.324754  0.9350295  1.038155
##   2        22      1.324754  0.9350295  1.038155
##   2        23      1.324754  0.9350295  1.038155
##   2        24      1.324754  0.9350295  1.038155
##   2        25      1.324754  0.9350295  1.038155
##   2        26      1.324754  0.9350295  1.038155
##   2        27      1.324754  0.9350295  1.038155
##   2        28      1.324754  0.9350295  1.038155
##   2        29      1.324754  0.9350295  1.038155
##   2        30      1.324754  0.9350295  1.038155
##   2        31      1.324754  0.9350295  1.038155
##   2        32      1.324754  0.9350295  1.038155
##   2        33      1.324754  0.9350295  1.038155
##   2        34      1.324754  0.9350295  1.038155
##   2        35      1.324754  0.9350295  1.038155
##   2        36      1.324754  0.9350295  1.038155
##   2        37      1.324754  0.9350295  1.038155
##   2        38      1.324754  0.9350295  1.038155
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 16 and degree = 2.
```

```r
plot(marsModel)
```



Which predictors did MARS select as the informative?

```r
varImp(marsModel)
```

```
## earth variable importance
##
##     Overall
## X1   100.00
## X4    85.14
## X2    69.24
## X5    49.32
## X3    40.02
## X9     0.00
## X8     0.00
## X10    0.00
## X7     0.00
## X6     0.00
```

Only X1-X5

Performance evaluation

```r
marsPred <- predict(marsModel, newdata = testData$x)
modelstats <- rbind(modelstats, postResample(pred = marsPred, obs = testData$y))
postResample(pred = marsPred, obs = testData$y)
```

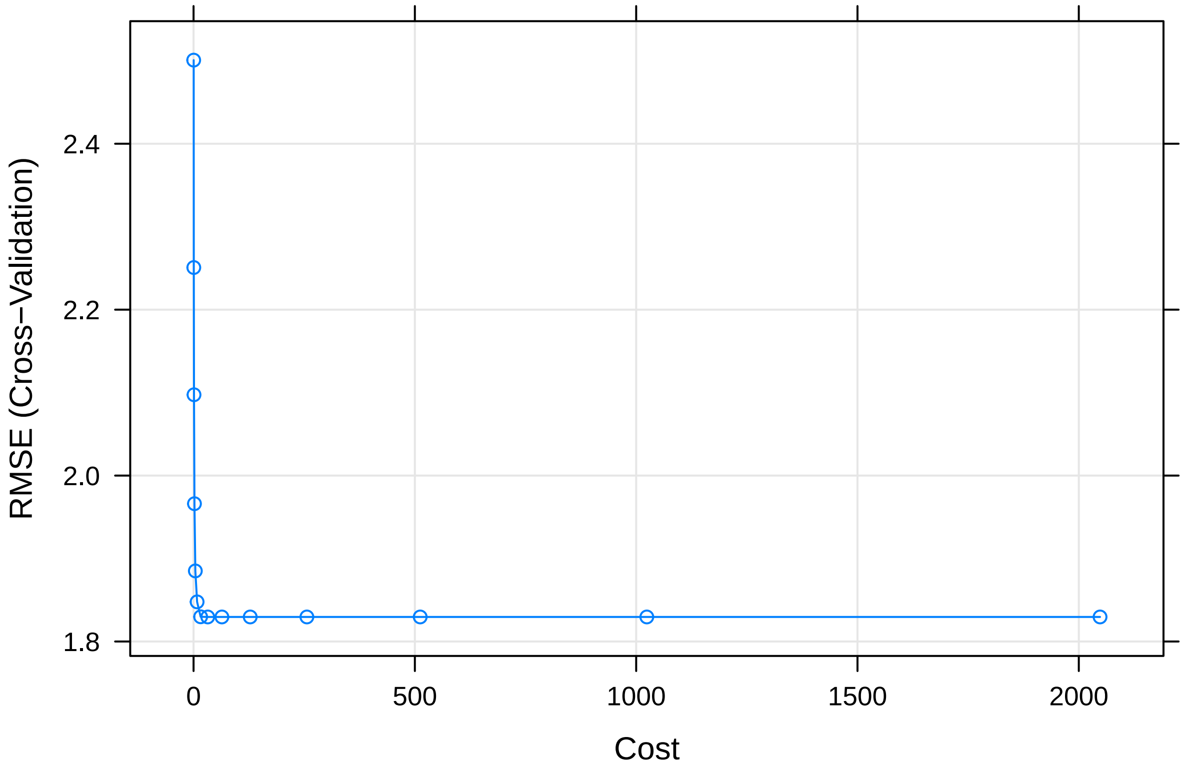
```
##      RMSE  Rsquared       MAE
## 1.1492504 0.9471145 0.9158382
```

SVM Radial

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

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictors
##
## 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  2.500756  0.7945828  2.015384
##      0.50  2.250771  0.8099288  1.806911
##      1.00  2.097422  0.8277370  1.668132
##      2.00  1.966109  0.8487748  1.547347
##      4.00  1.885043  0.8587663  1.486123
##      8.00  1.847760  0.8642743  1.452813
##     16.00  1.829788  0.8682447  1.443874
##     32.00  1.829579  0.8685594  1.444677
##     64.00  1.829579  0.8685594  1.444677
##    128.00  1.829579  0.8685594  1.444677
##    256.00  1.829579  0.8685594  1.444677
##    512.00  1.829579  0.8685594  1.444677
##   1024.00  1.829579  0.8685594  1.444677
##   2048.00  1.829579  0.8685594  1.444677
##
## Tuning parameter 'sigma' was held constant at a value of 0.05494384
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.05494384 and C = 32.
```

```r
plot(svmRModel)
```




```r
svmRModel$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 =  0.0549438430536528
##
## Number of Support Vectors : 152
##
## Objective Function Value : -90.8443
## Training error : 0.008371
```

Performance evaluation

```r
svmRPred <- predict(svmRModel, newdata = testData$x)
modelstats <- rbind(modelstats, postResample(pred = svmRPred, obs = testData$y))
postResample(pred = svmRPred, obs = testData$y)
```

```
##      RMSE  Rsquared       MAE
## 2.0574214 0.8283777 1.5642193
```

Which model performs the best?

```r
rownames(modelstats) <- c('NNET', 'KNN', 'MARS', 'SVM')
modelstats
```

```
##          RMSE  Rsquared       MAE
## NNET 1.926316 0.8528292 1.4630365
## KNN  3.204059 0.6819919 2.5683461
## MARS 1.149250 0.9471145 0.9158382
## SVM  2.057421 0.8283777 1.5642193
```

MARS model gives the best performance. Also it selects only informative predictors (X1-X5)