Exercise 4

Austin Vanderlyn ajl745

7/24/2022

Exercise 4

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

Tune several models on these data. Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)? Libraries;

```
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version 4.1.3
```

```
library(caret)
```

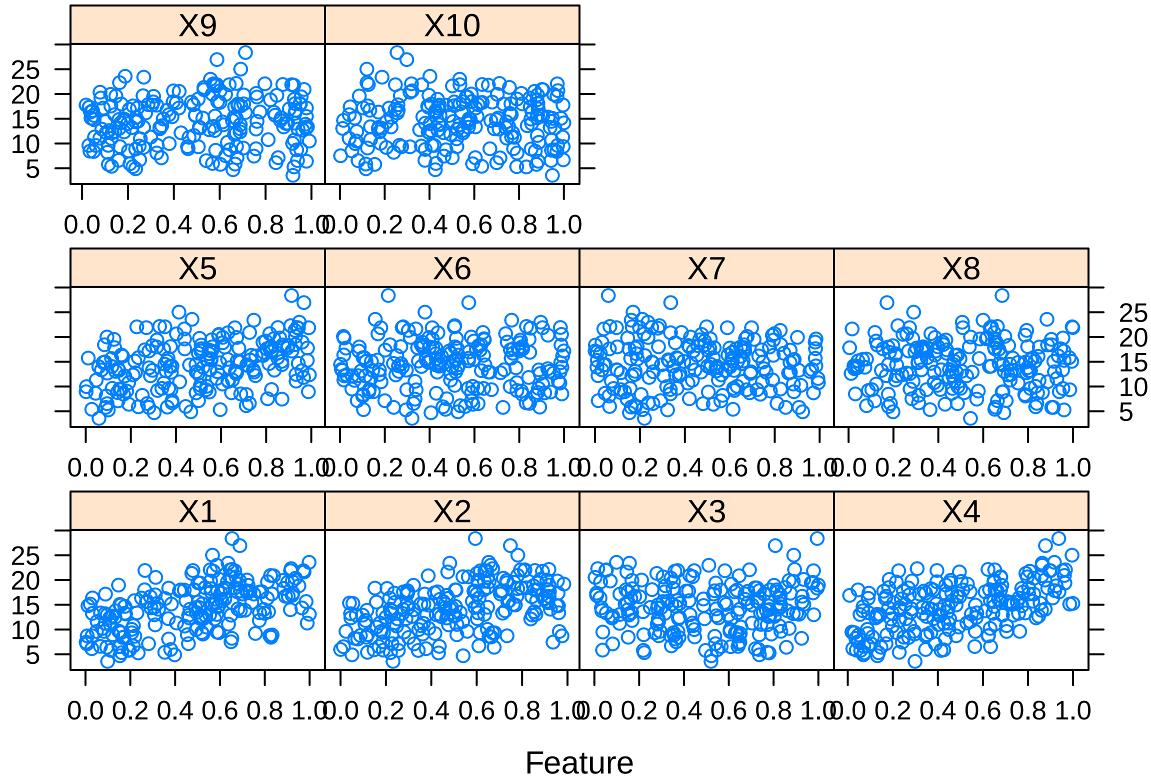
```
## Warning: package 'caret' was built under R version 4.1.2

## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 4.1.3
```

Load data;

```
set.seed(200)
trainingData = mlbench.friedman1(200, sd = 1)
trainingData$x = data.frame(trainingData$x)
testData = mlbench.friedman1(5000, sd = 1)
testData$x = data.frame(testData$x)
featurePlot(trainingData$x, trainingData$y)
```



The first model is already posted in the problem, a Knn model, but I will rebuild it here to get the values for comparison.

KNN Model

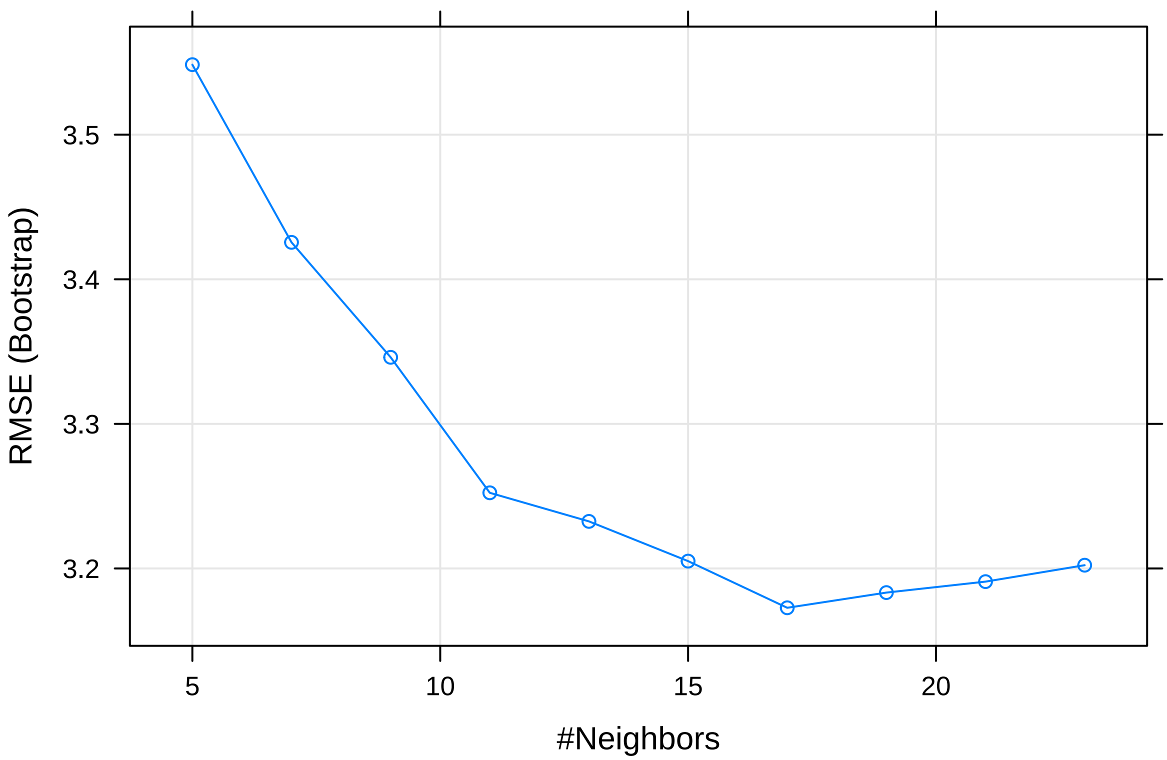
Tune KNN model;

```
set.seed(123)
knnTune = train(trainingData$x, trainingData$y,
                method = "knn",
                preProc = c("center", "scale"),
                tuneLength = 10)
knnTune
```

```
## 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  3.548433  0.4919564  2.888283
##     7  3.425531  0.5255725  2.778090
##     9  3.346026  0.5523023  2.704791
##    11  3.252313  0.5875603  2.620492
##    13  3.232552  0.6000482  2.601113
##    15  3.205067  0.6203296  2.586704
##    17  3.172791  0.6408339  2.566738
##    19  3.183306  0.6494300  2.587220
##    21  3.190873  0.6556293  2.596793
##    23  3.202234  0.6597746  2.604279
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 17.
```

```
plot(knnTune)
```



KNN predictions;

```
testResults = data.frame(obs = testData,
                         knn = predict(knnTune, testData$x))
postResample(pred = predict(knnTune, testData$x), testData$y)
```

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

MARS Model

Fit MARS model;

```
set.seed(123)
marsGrid = expand.grid(degree = 1,
                       nprune = 2:38)

marsTune = train(trainingData$x, trainingData$y,
                 method = "earth",
                 preProc = c("center", "scale"),
                 tuneGrid = marsGrid)
```

```
## Loading required package: earth

## Warning: package 'earth' was built under R version 4.1.3

## Loading required package: Formula

## Loading required package: plotmo

## Warning: package 'plotmo' was built under R version 4.1.3

## Loading required package: plotrix

## Loading required package: TeachingDemos

## Warning: package 'TeachingDemos' was built under R version 4.1.3
```

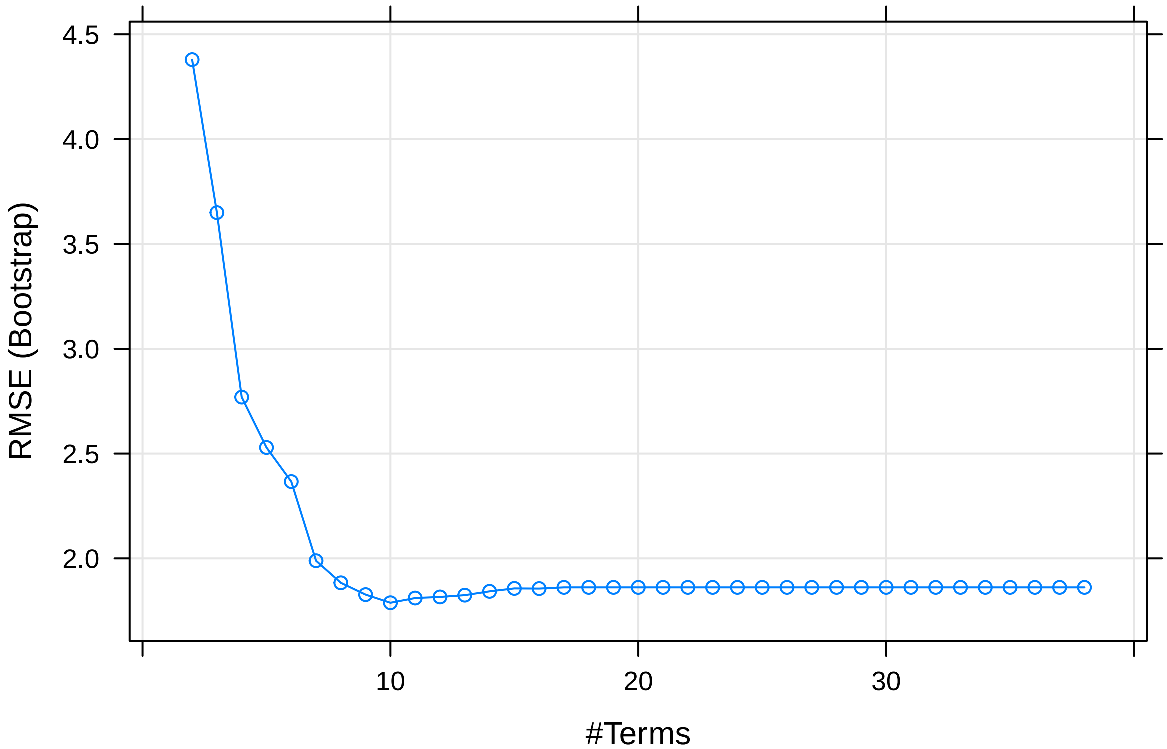
```
marsTune
```

```
## Multivariate Adaptive Regression Spline
##
## 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:
##
##   nprune  RMSE      Rsquared   MAE
##    2      4.379381  0.2301740  3.575902
##    3      3.649438  0.4583683  2.944879
##    4      2.769352  0.6876944  2.223704
##    5      2.529007  0.7399204  2.018331
##    6      2.366383  0.7734368  1.888582
##    7      1.988717  0.8380231  1.581362
##    8      1.883468  0.8556729  1.484933
##    9      1.827116  0.8637619  1.443208
##   10      1.788268  0.8690065  1.410531
```

```
##   11       1.810724  0.8662722  1.424227
##   12       1.815936  0.8656814  1.422587
##   13       1.824463  0.8644827  1.433229
##   14       1.842692  0.8615394  1.450268
##   15       1.856755  0.8590033  1.460392
##   16       1.855987  0.8591456  1.456274
##   17       1.861692  0.8581446  1.459553
##   18       1.861692  0.8581446  1.459553
##   19       1.861692  0.8581446  1.459553
##   20       1.861692  0.8581446  1.459553
##   21       1.861692  0.8581446  1.459553
##   22       1.861692  0.8581446  1.459553
##   23       1.861692  0.8581446  1.459553
##   24       1.861692  0.8581446  1.459553
##   25       1.861692  0.8581446  1.459553
##   26       1.861692  0.8581446  1.459553
##   27       1.861692  0.8581446  1.459553
##   28       1.861692  0.8581446  1.459553
##   29       1.861692  0.8581446  1.459553
##   30       1.861692  0.8581446  1.459553
##   31       1.861692  0.8581446  1.459553
##   32       1.861692  0.8581446  1.459553
##   33       1.861692  0.8581446  1.459553
##   34       1.861692  0.8581446  1.459553
##   35       1.861692  0.8581446  1.459553
##   36       1.861692  0.8581446  1.459553
##   37       1.861692  0.8581446  1.459553
##   38       1.861692  0.8581446  1.459553
##
## Tuning parameter 'degree' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 10 and degree = 1.
```

Plot;

```
plot(marsTune)
```



Mars predictions;

```
testResults$mars = predict(marsTune, testData$x)

postResample(pred = predict(marsTune, testData$x), testData$y)
```

```
##     RMSE Rsquared      MAE
## 1.776575 0.872700 1.358367
```

SVM Model

Fit SVM Radial model;

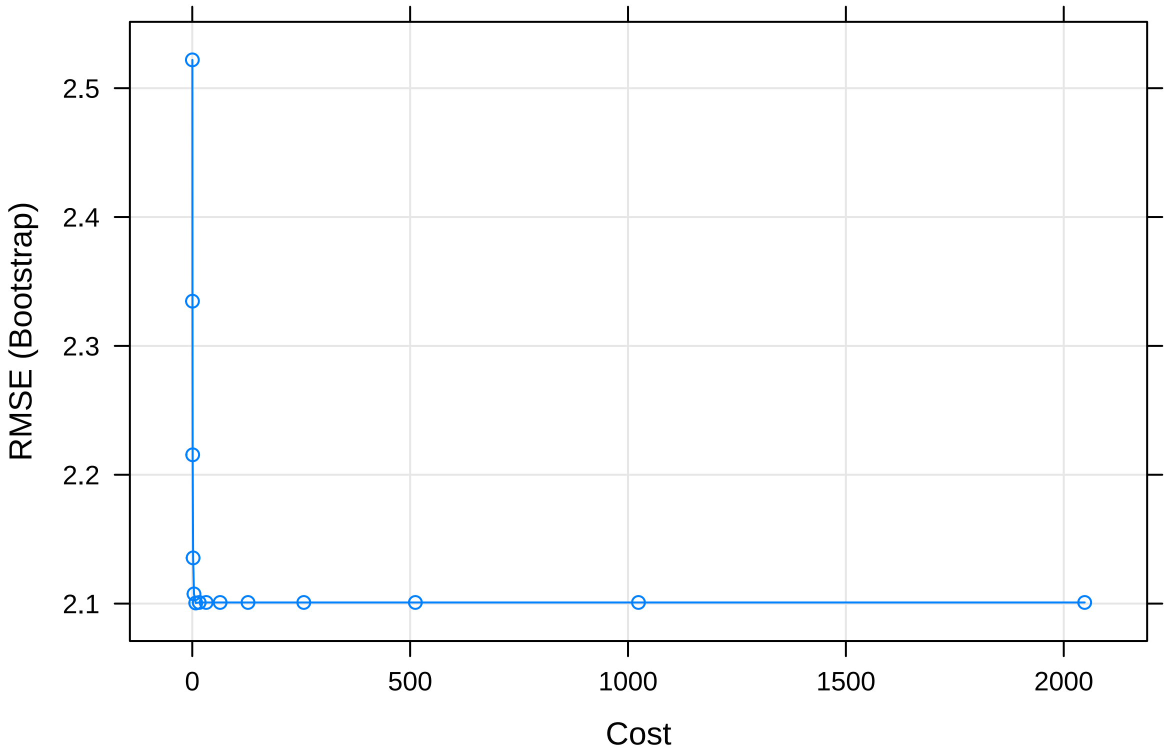
```
set.seed(123)
svmRTune = train(trainingData$x, trainingData$y,
                 method = "svmRadial",
                 preProc = c("center", "scale"),
                 tuneLength = 14)
svmRTune
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 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:
##
##   C        RMSE      Rsquared   MAE
##      0.25  2.521970  0.7754171  2.000615
##      0.50  2.334701  0.7878369  1.837681
##      1.00  2.215481  0.8032030  1.738507
##      2.00  2.135470  0.8138612  1.679711
##      4.00  2.107537  0.8177454  1.651024
##      8.00  2.100487  0.8189859  1.648012
##     16.00  2.100901  0.8189207  1.648818
##     32.00  2.100901  0.8189207  1.648818
##     64.00  2.100901  0.8189207  1.648818
##    128.00  2.100901  0.8189207  1.648818
##    256.00  2.100901  0.8189207  1.648818
##    512.00  2.100901  0.8189207  1.648818
##   1024.00  2.100901  0.8189207  1.648818
##   2048.00  2.100901  0.8189207  1.648818
##
## Tuning parameter 'sigma' was held constant at a value of 0.06510592
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.06510592 and C = 8.
```

Plot;

```
plot(svmRTune)
```



SVM Radial predictions;

```
testResults$svmR = predict(svmRTune, testData$x)

postResample(pred = predict(svmRTune, testData$x), testData$y)
```

```
##      RMSE  Rsquared       MAE
## 2.0634091 0.8275351 1.5663529
```

Results

```
mars.pred = predict(marsTune, testData$x)
svmR.pred = predict(svmRTune, testData$x)
knn.pred = predict(knnTune, testData$x)

test = data.frame(rbind(MARS = postResample(pred = mars.pred, obs = testData$y),
                 SVMr = postResample(pred = svmR.pred, obs = testData$y),
                 KNN = postResample(pred = knn.pred, obs = testData$y)))
test
```

```
##          RMSE  Rsquared      MAE
## MARS 1.776575 0.8727000 1.358367
## SVMr 2.063409 0.8275351 1.566353
## KNN  3.204059 0.6819919 2.568346
```

We can see the MARS is clearly the best model, with the lowest RMSE and MAE and the highest R^2.

Now to the question of whether MARS selects the important predictors X1:X5.

```
summary(marsTune)
```

```
## Call: earth(x=data.frame[200,10], y=c(18.46,16.1,17...), keepxy=TRUE, degree=1,
##             nprune=10)
##
##                  coefficients
## (Intercept)        20.3958041
## h(0.507267-X1)     -3.0209971
## h(0.325504-X2)     -2.8963069
## h(X3- -0.804171)    1.1187319
## h(-0.216741-X3)     3.4950111
## h(X3-0.453446)      2.1548596
## h(0.953812-X4)     -2.7559239
## h(X4-0.953812)      2.8600536
## h(1.17878-X5)      -1.5056208
## h(X6- -0.47556)    -0.5025995
##
## Selected 10 of 18 terms, and 6 of 10 predictors (nprune=10)
## Termination condition: Reached nk 21
## Importance: X1, X4, X2, X5, X3, X6, X7-unused, X8-unused, X9-unused, ...
## Number of terms at each degree of interaction: 1 9 (additive model)
## GCV 2.731203    RSS 447.3848    GRSq 0.889112    RSq 0.9082649
```

Under Importance we can see that the Mars model does indeed select X1 through X5, and doesn't use X6 through X10.

```
varImp(marsTune)
```

```
## earth variable importance
##
##    Overall
## X1  100.00
## X4   82.78
## X2   64.18
## X5   40.21
## X3   28.14
## X6    0.00
```

```
plot(varImp(marsTune))
```