Data624 - Homework8

Amit Kapoor

4/19/2021

Contents


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

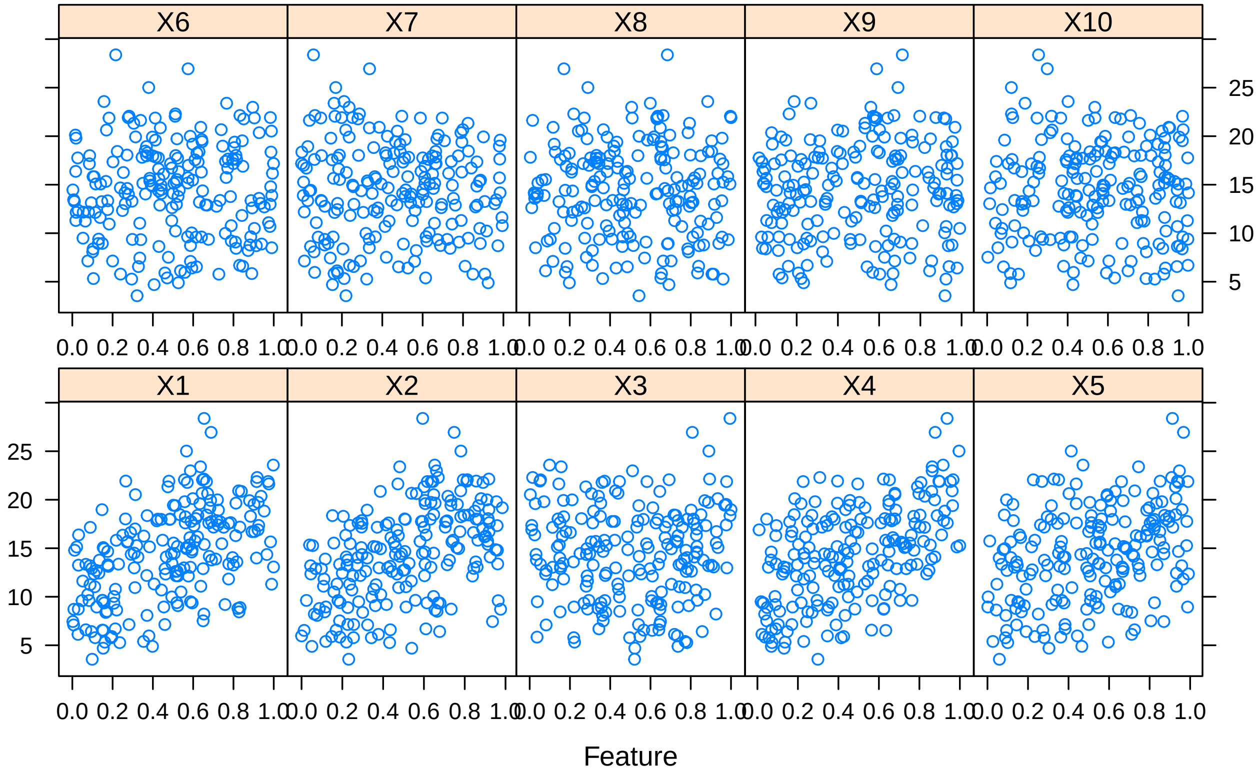
Exercise 7.2

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)
trainingData$x <- data.frame(trainingData$x)
# featurePlot
featurePlot(trainingData$x, trainingData$y)
```




```r
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~
```

```r
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

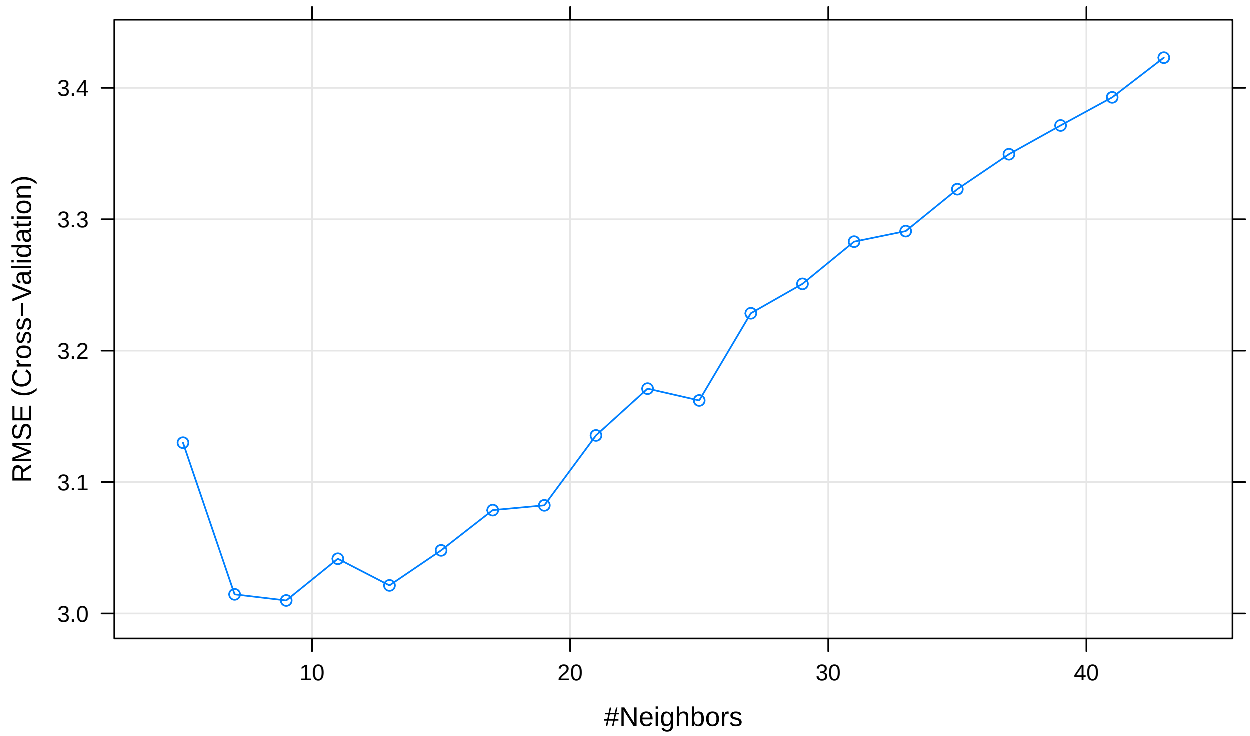
```r
set.seed(317)
knnfit <- train(trainingData$x,
                trainingData$y,
                method = "knn",
                preProcess = c("center","scale"),
                tuneLength = 20,
                trControl = trainControl(method = "cv"))

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.
```

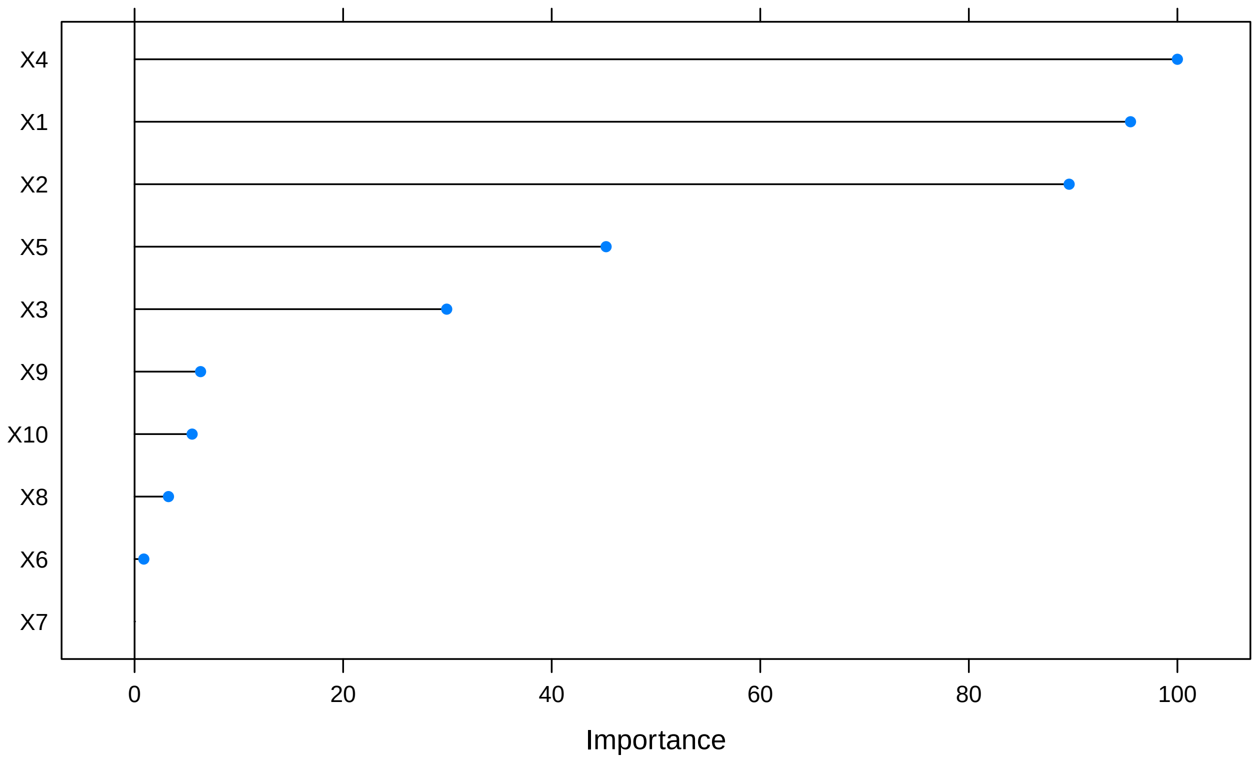
```
knnfit$bestTune
```

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

```
plot(knnfit)
```




```
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
```

Support Vector Machines

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

svmfit
```

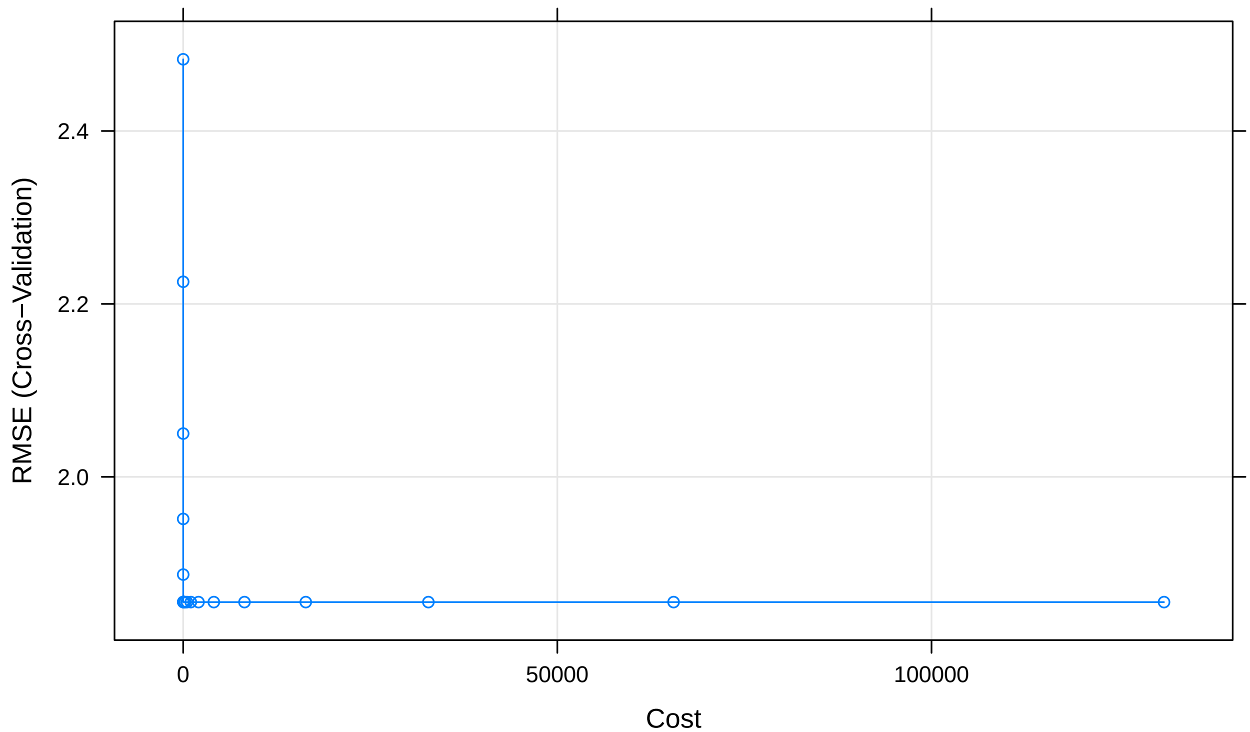
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##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

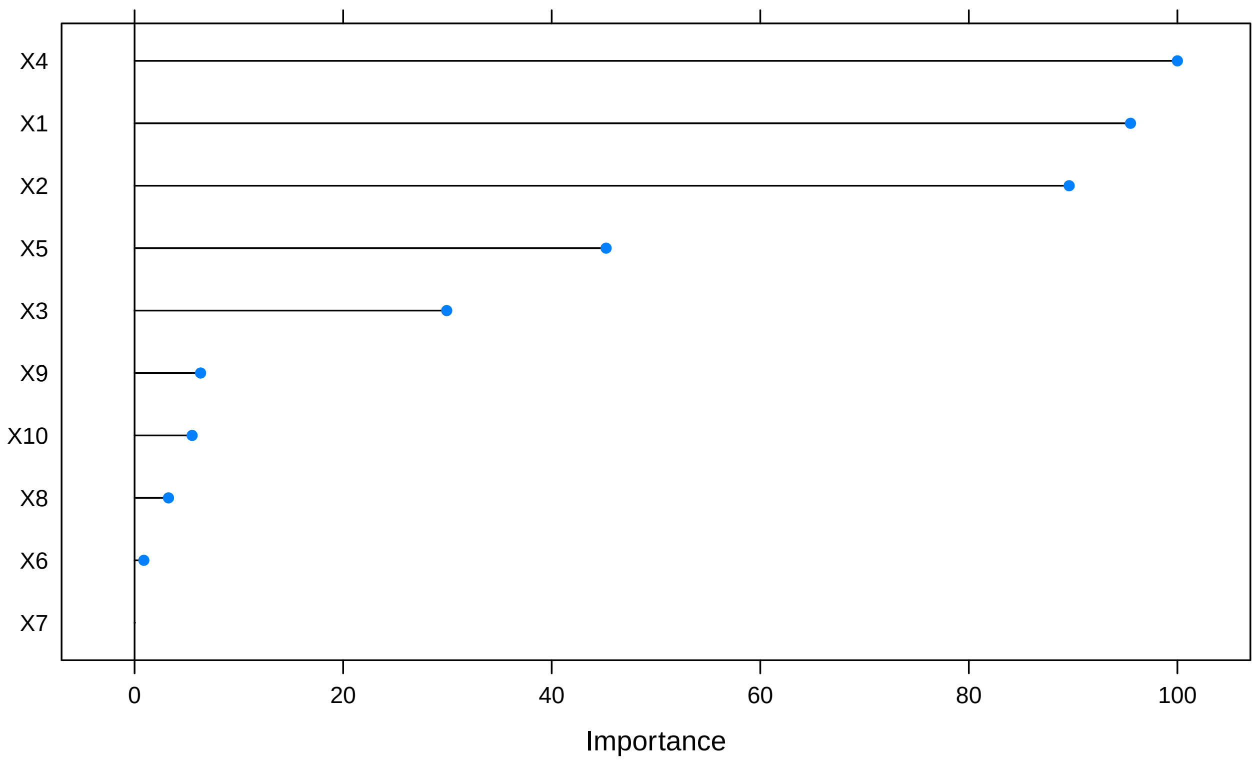
```
svmfit$finalModel
```

```
## 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(svmfit)
```




```r
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
```

Multivariate Adaptive Regression Splines

```r
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
```

```r
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
```

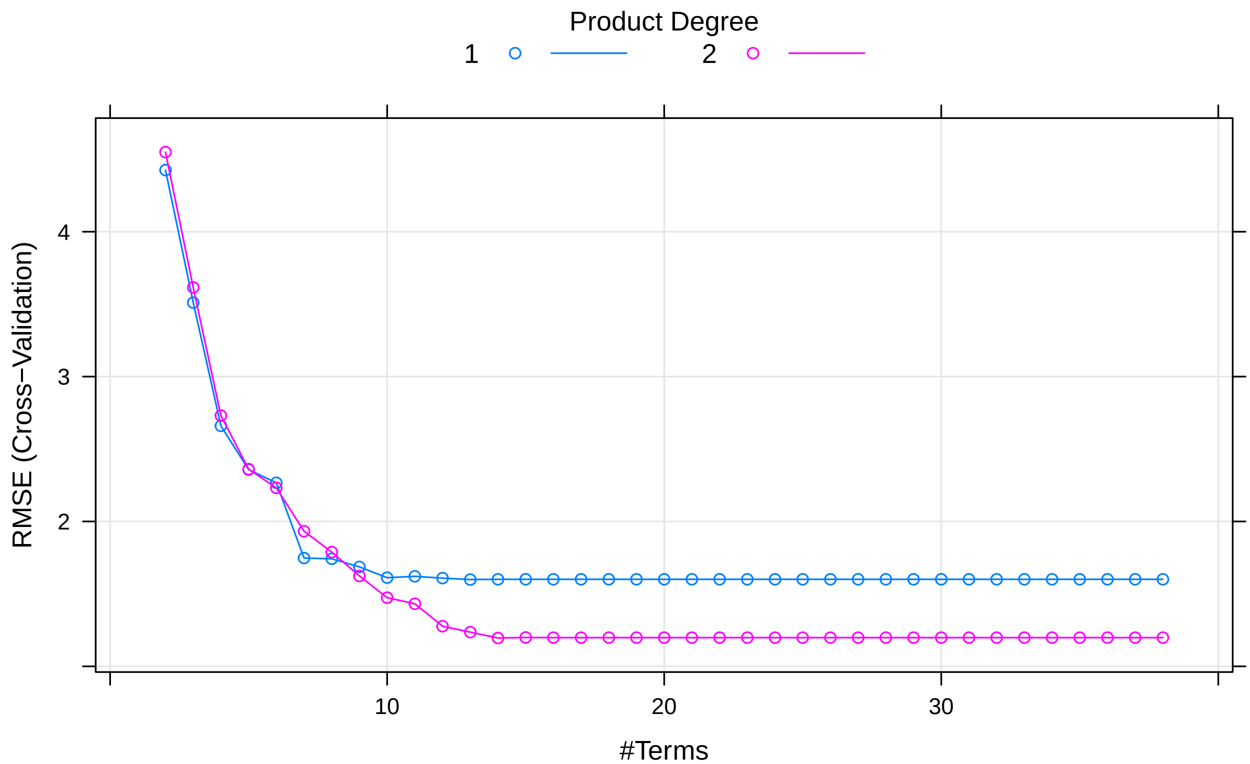
```
## 1       23      1.600854  0.8985110  1.2482796
## 1       24      1.600854  0.8985110  1.2482796
## 1       25      1.600854  0.8985110  1.2482796
## 1       26      1.600854  0.8985110  1.2482796
## 1       27      1.600854  0.8985110  1.2482796
## 1       28      1.600854  0.8985110  1.2482796
## 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.
```

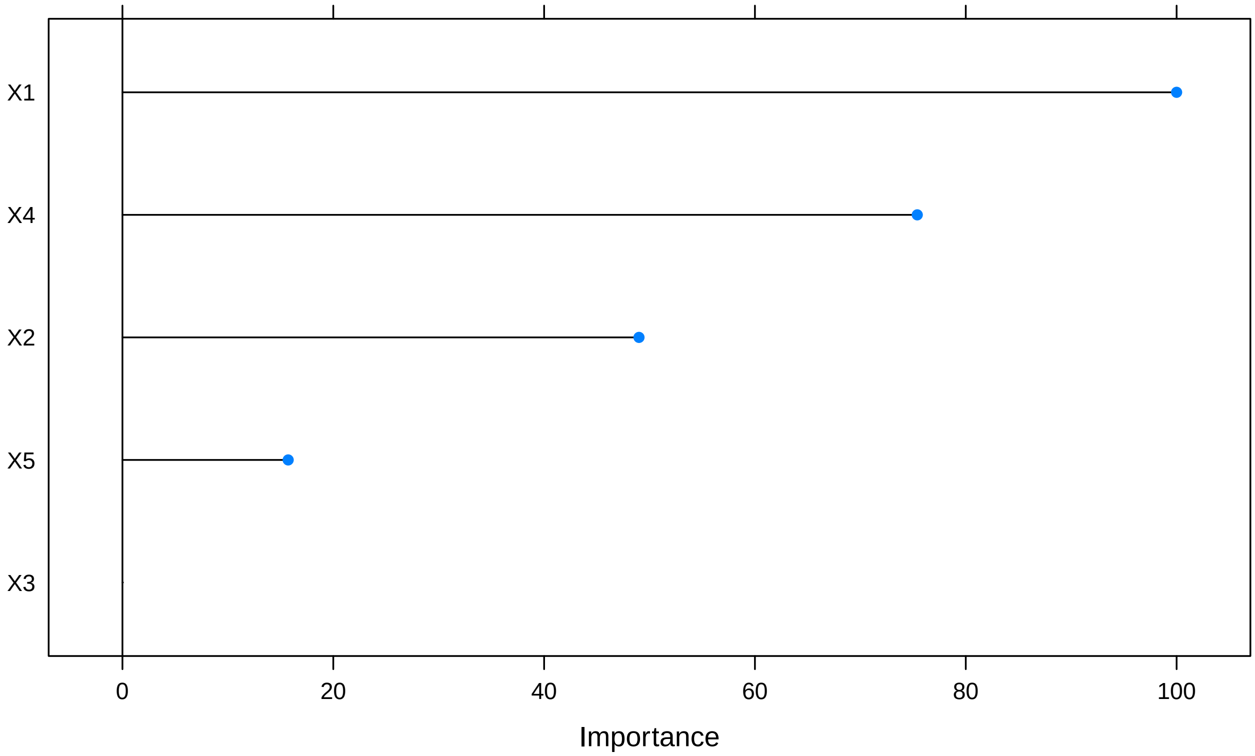
```r
marsfit$bestTune
```

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

```r
plot(marsfit)
```




```r
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
```

Neural Networks

```
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)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
nnetfit
```

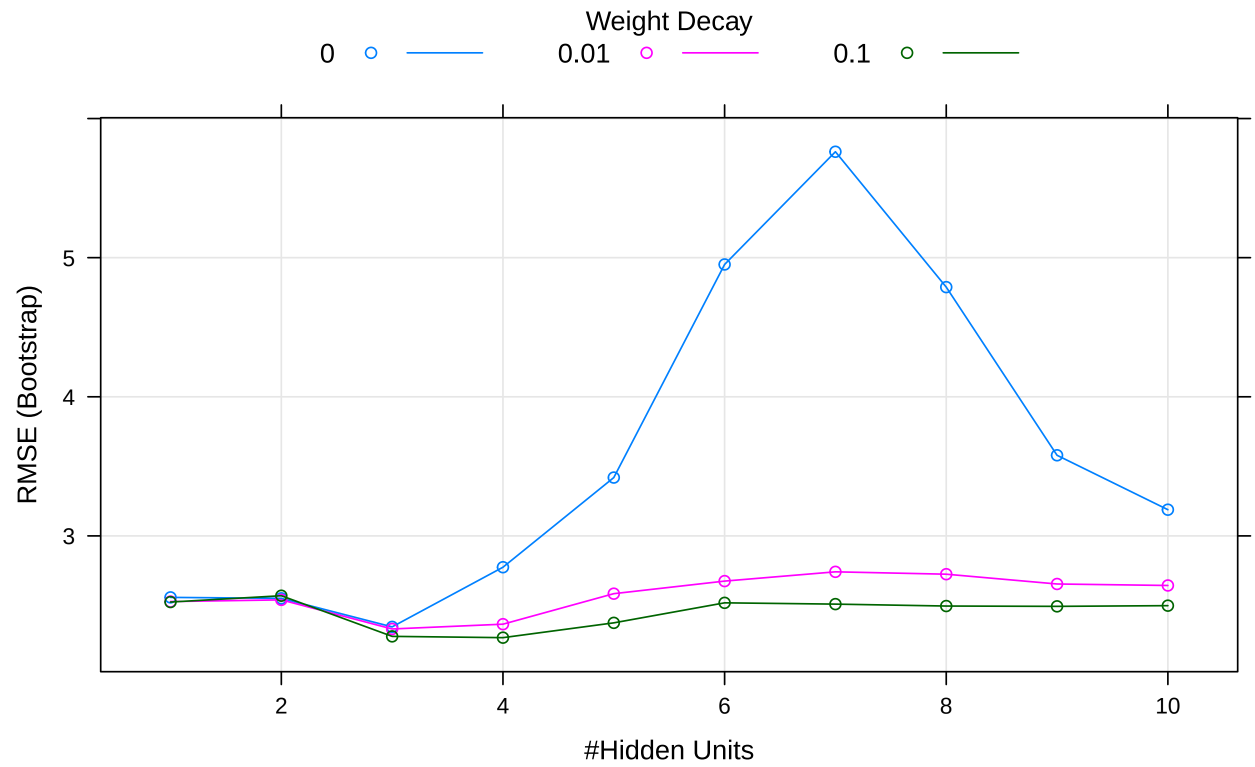
```
## Model Averaged Neural Network
##
## 200 samples
##  10 predictor
```

```
##
## 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.
```

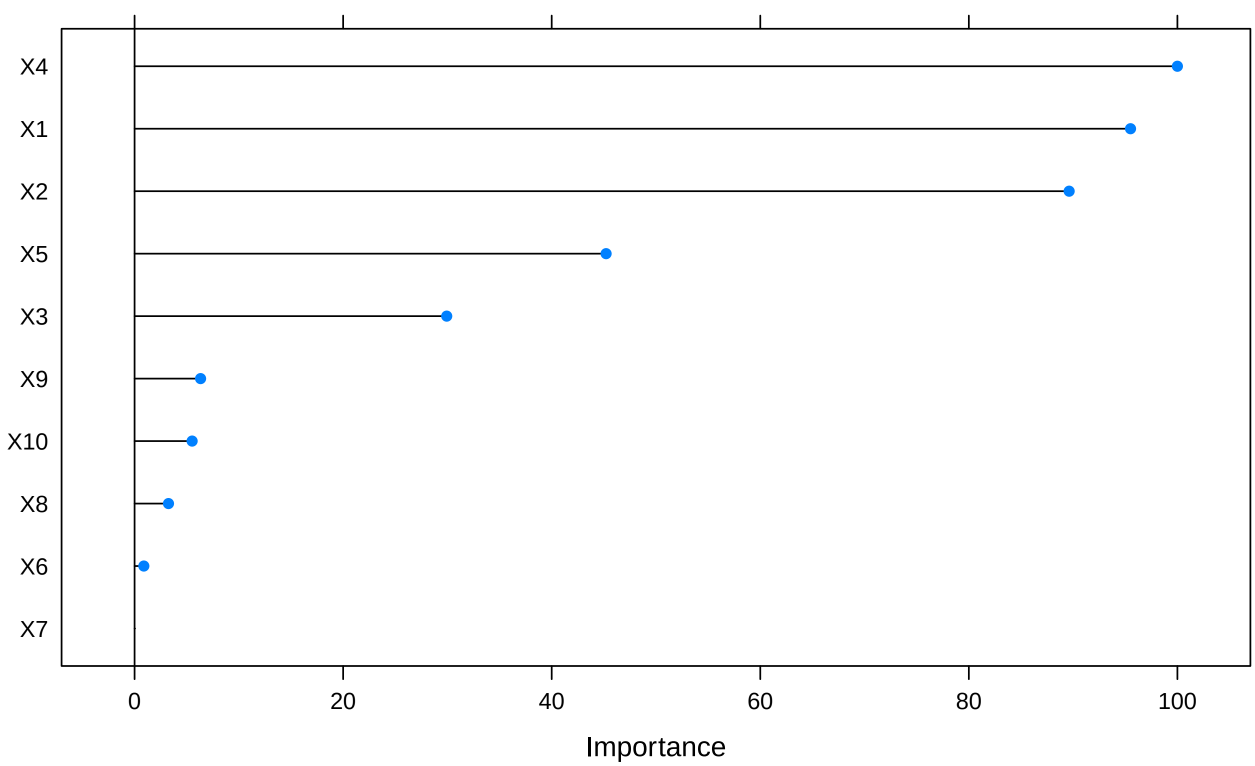
```
nnetfit$bestTune
```

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

```
plot(nnetfit)
```




```
plot(varImp(nnetfit))
```




```
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
```

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
```

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

(a)

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

(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?

(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?