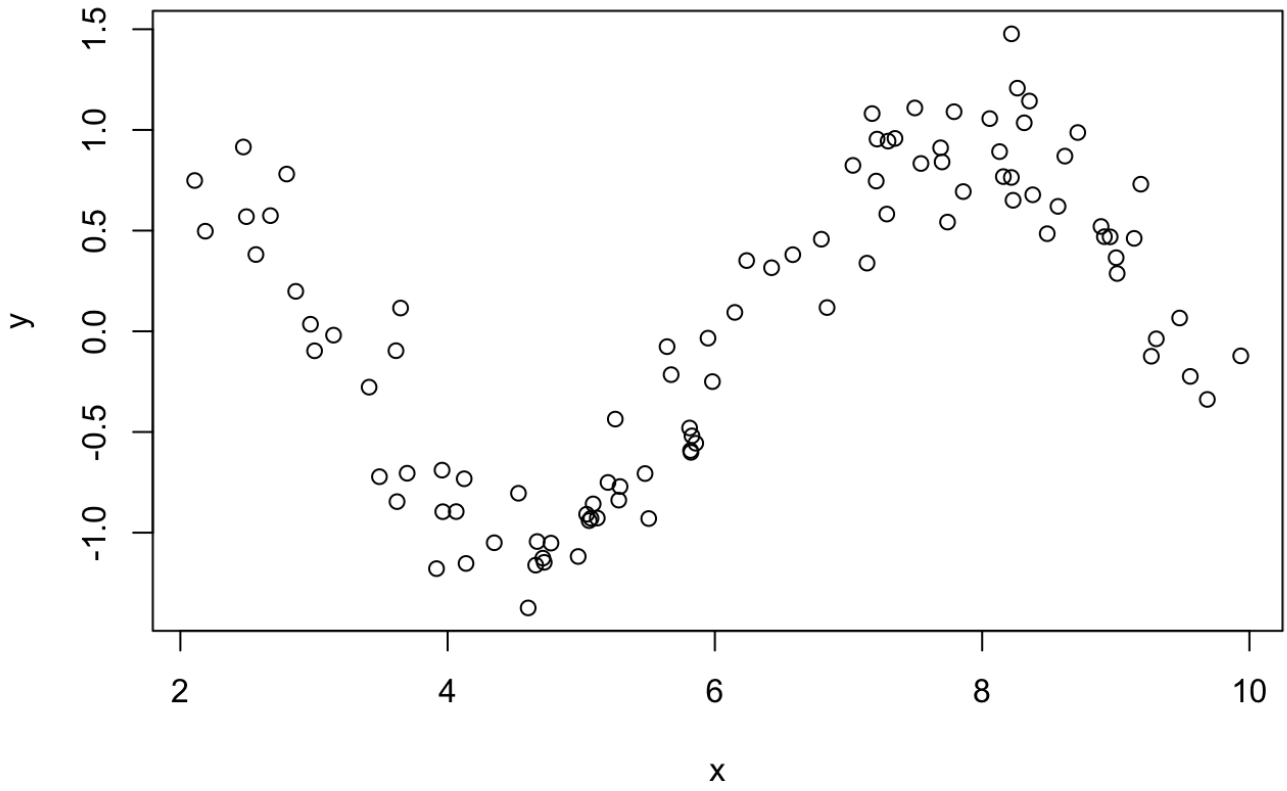
Assignment 4

Nishanth Gandhidoss

11/02/2017

Question 1

For this question, we have given with some code that could generate sin curve with some noise. Lets use plot the data below and store that the data in a dataframe for future use. Also I have displayed the top five data points in the data

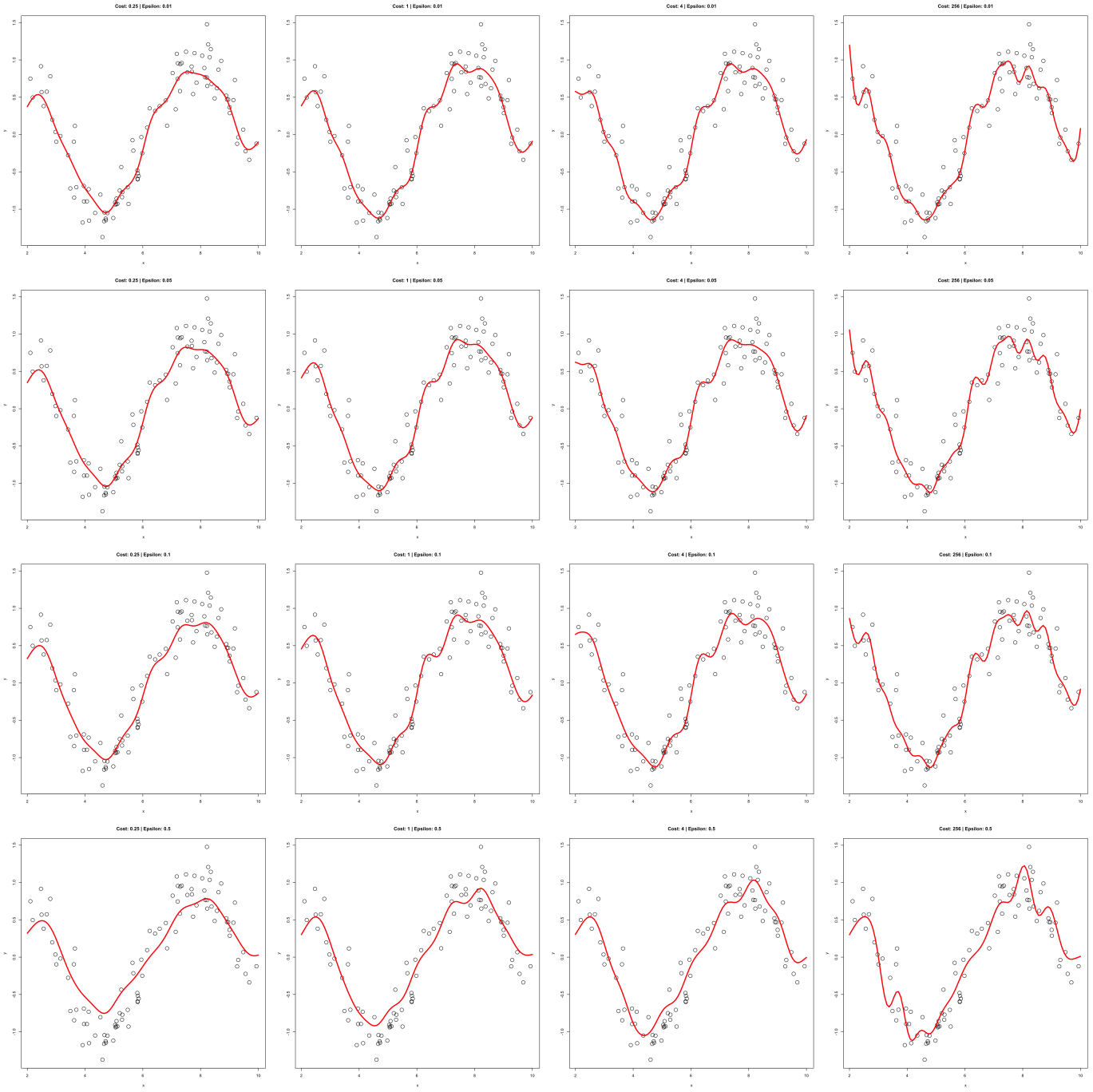



```
##          x
## 1 2.000000
## 2 2.080808
## 3 2.161616
## 4 2.242424
## 5 2.323232
## 6 2.404040
```

Section (a)

As per the instructions, I am using a seed(1) in order to have reporduciability of my code. The following table shows the parameters values that I have used to fit the different models using radial basis functions.

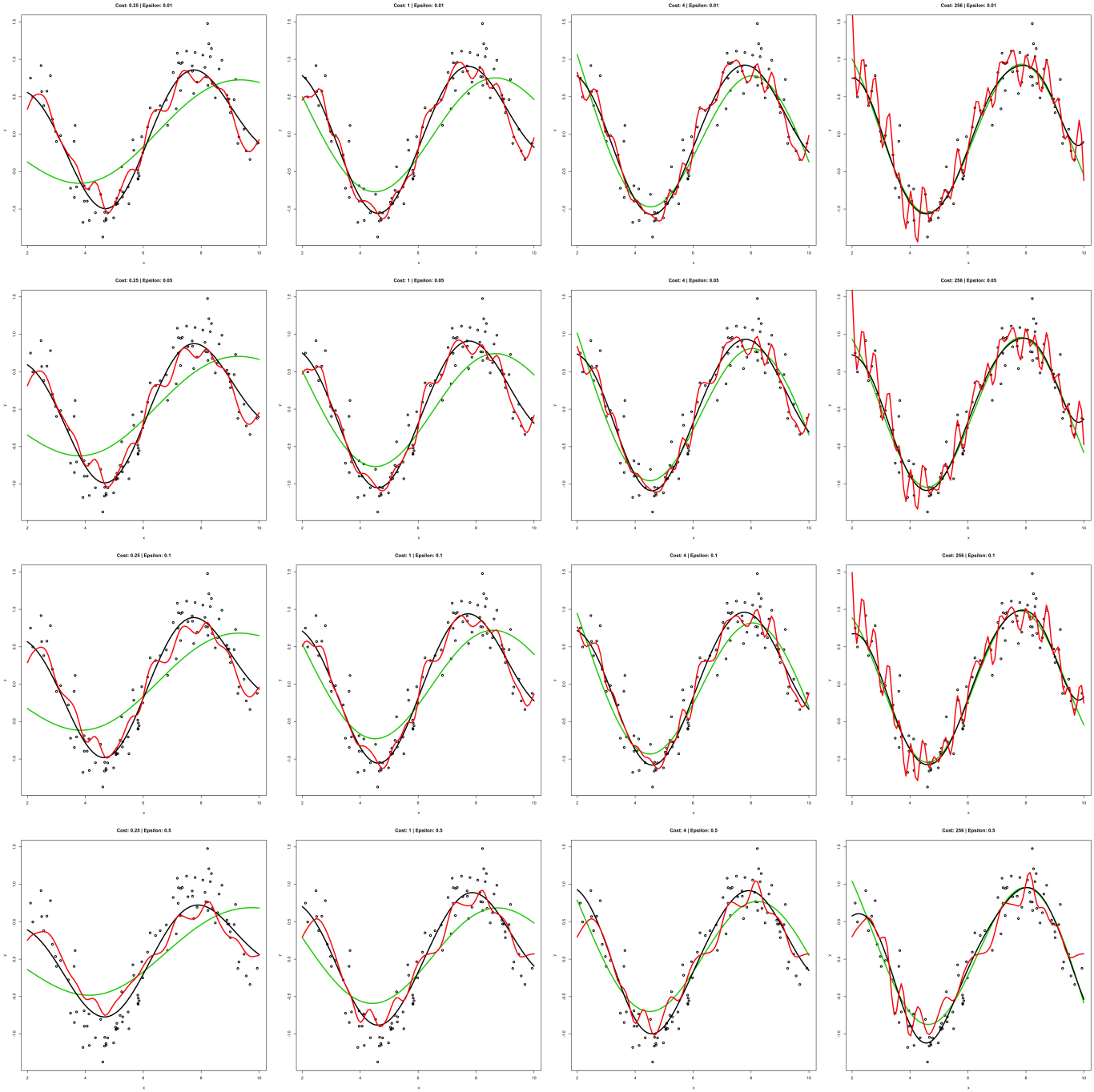
```
##      costs epsilons
## 1     0.25      0.01
## 2     1.00      0.01
## 3     4.00      0.01
## 4   256.00      0.01
## 5     0.25      0.05
## 6     1.00      0.05
## 7     4.00      0.05
## 8   256.00      0.05
## 9     0.25      0.10
## 10    1.00      0.10
## 11    4.00      0.10
## 12  256.00      0.10
## 13    0.25      0.50
## 14    1.00      0.50
## 15    4.00      0.50
## 16  256.00      0.50
```

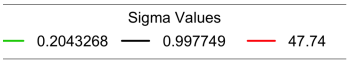


Thus above we have the plot for various cost and epsilon values parameter fitted over the given curve.

Section (b)

Now lets add the sigma parameter as a hyperparamer. Each lines shows different sigma values with different colors.





Cost

From the plot, we can see that increasing the cost will increase the model complexity. Thus model with low cost will have high bias while the model with high cost will have high variance.

Epsilon

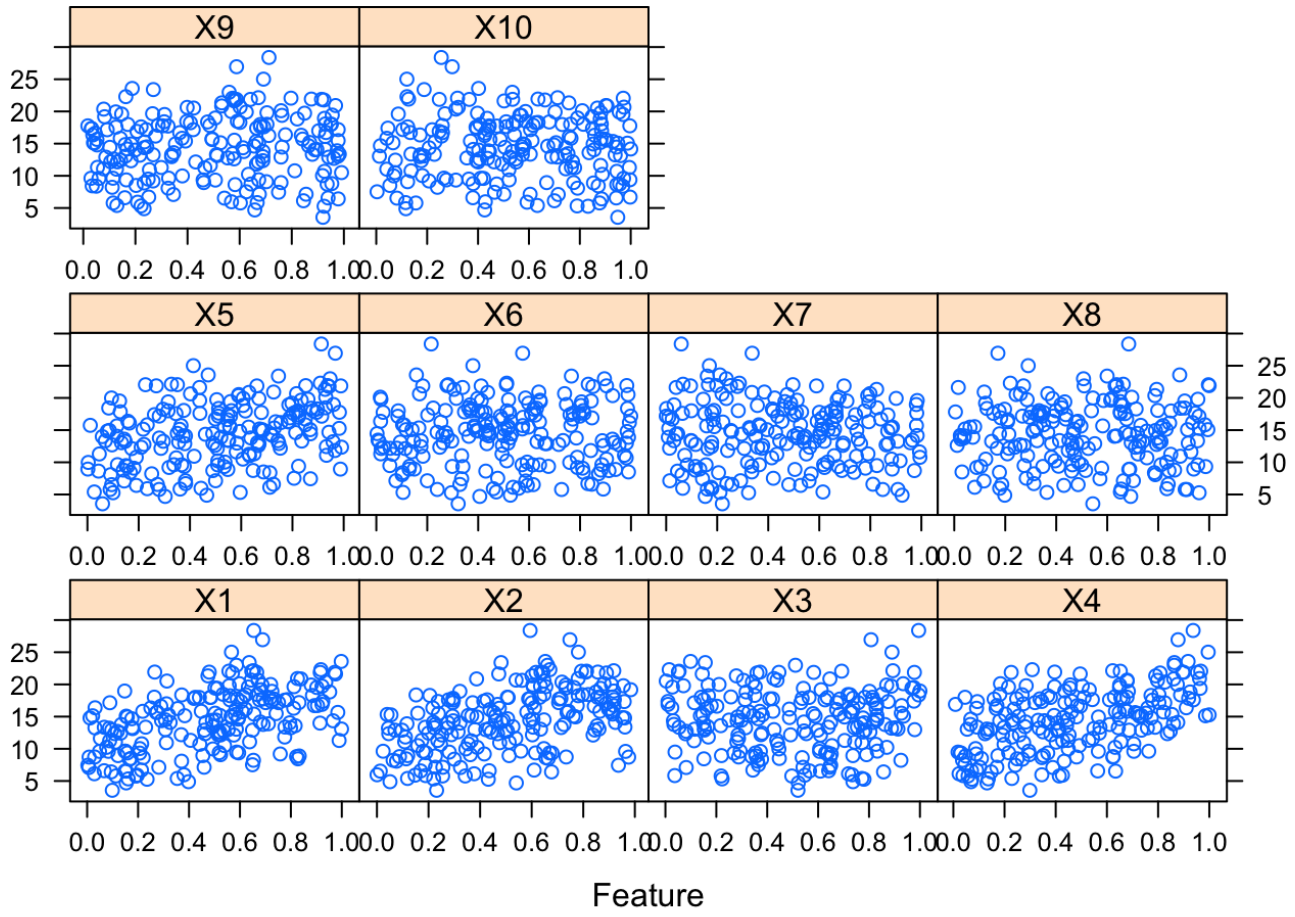
From the above plot in the last column, we can see that increasing the epsilon value decreases that model complexity. The plot on the upper right corner has cost of 256 and epsilon of 0.91 with high sigma value is shown in red color. It looks overfitting the data but the plot in the bottom of the same columnn is having less complex model compared tmo the top one. We can say that increasing the epsilon will make the model smooth.

Sigma

The increase in the sigma values is overfiting the data. We can clearly see that the green line with low sigma value is having a very simple fit(high bias) compared to red line with very complex fit(high variance).

Question 2

For this question we are going to work on the Friedman's dataset created by a simulation. mlbench package in R provides this data so that we can get it and use it. I going to load the train and test set of the data as it was directed in the book.

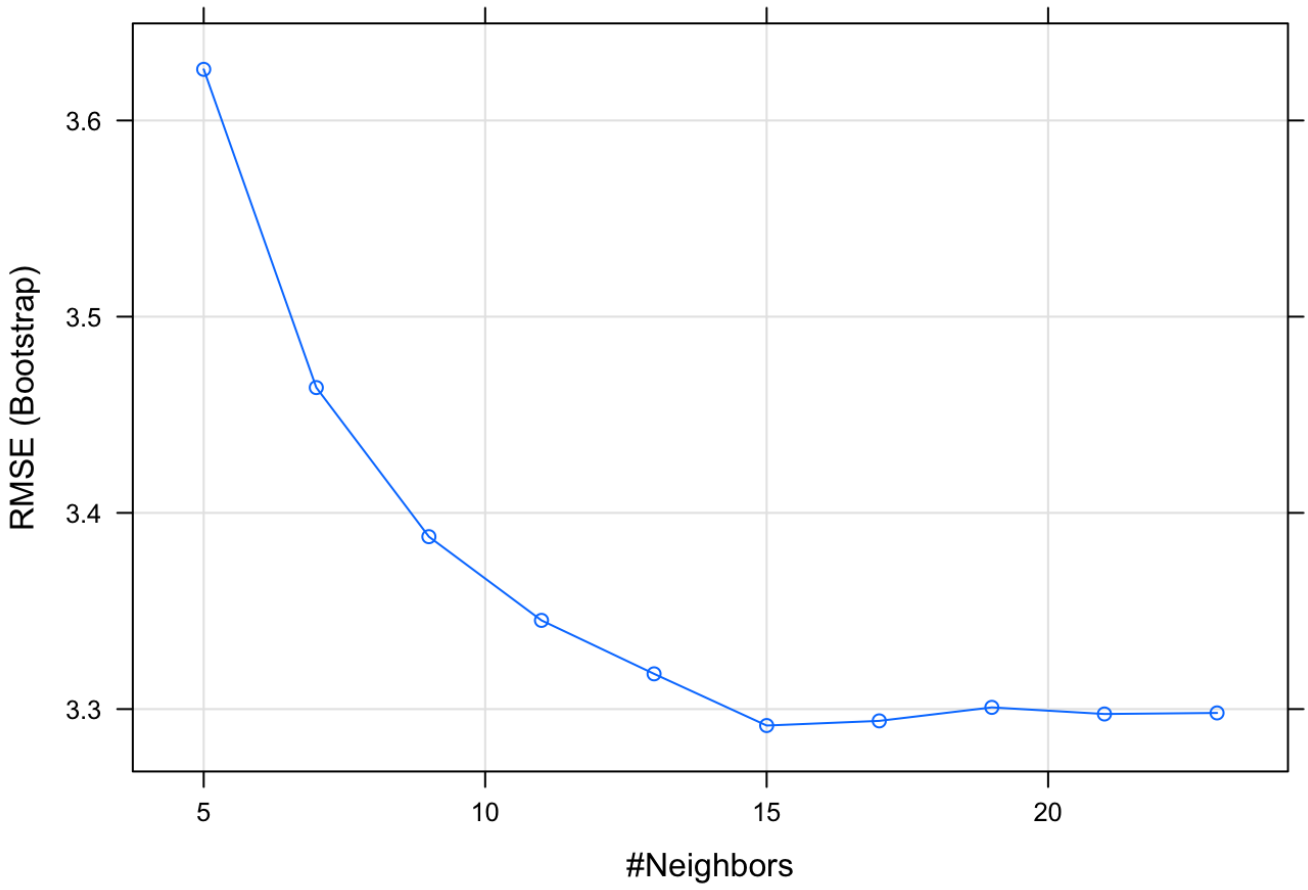


Thus the above plot shows that we have succesfully loaded the train and test set.

KNN

As given in the book, lets first fit the data with a KNN model. For this I have used train() in the caret package with knn as method. Below we have complete information about the KNN model.

```
## 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  3.626071  0.4766532  2.954620
##    7  3.463866  0.5231142  2.823065
##    9  3.387872  0.5552706  2.744969
##   11  3.345212  0.5754355  2.706971
##   13  3.317962  0.5964912  2.696927
##   15  3.291614  0.6123880  2.668077
##   17  3.293991  0.6238074  2.665458
##   19  3.300870  0.6342793  2.668902
##   21  3.297502  0.6472145  2.674792
##   23  3.298045  0.6534698  2.676965
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was k = 15.
```



Test set

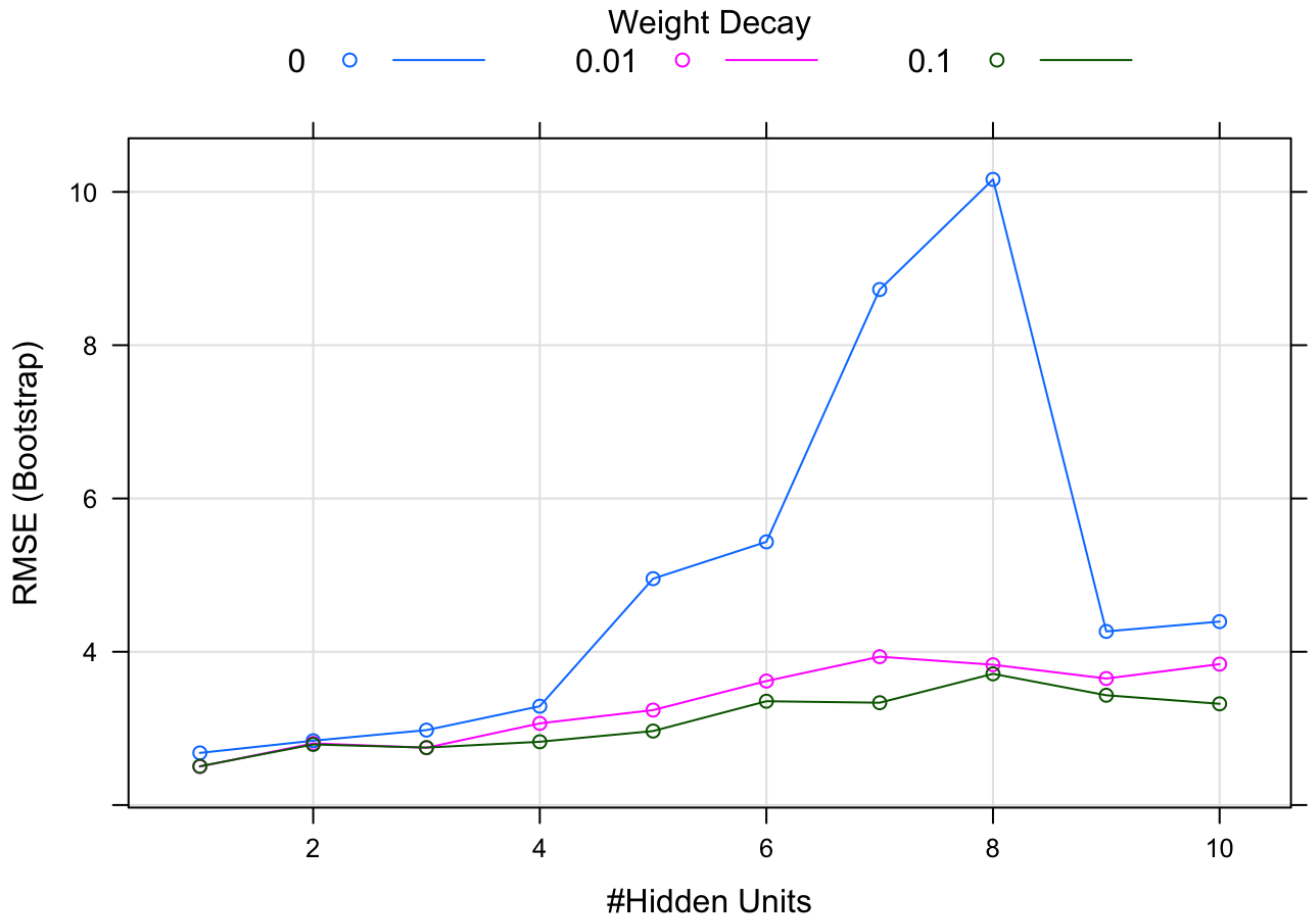
```
##      RMSE  Rsquared       MAE
## 3.1750657 0.6785946 2.5443169
```

From the above plot and the tabulated result, we can clearly see that the best model has the k value of 15. That is the best R^2 is obtain when we are considering 15 nearest neighbours. The R^2 value obtained on the test set is 0.6786.

Neural Network

Lets bulid the model using neural network model using train() in caret package with method as nnet. I am setting the decay to be 0, 0.01, 0.1 and size varying from 1 to 10.

```
## 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     2.680325  0.7141610  2.098819
##   0.00    2     2.839397  0.6861113  2.235487
##   0.00    3     2.977969  0.6701142  2.341324
##   0.00    4     3.289448  0.6282245  2.501787
##   0.00    5     4.953705  0.4986515  3.165885
##   0.00    6     5.434198  0.4491815  3.316184
##   0.00    7     8.726366  0.2937632  4.601882
##   0.00    8    10.163375  0.3888125  5.030177
##   0.00    9     4.265547  0.5062168  3.216742
##   0.00   10     4.393696  0.4964493  3.338989
##   0.01    1     2.502881  0.7525307  1.935655
##   0.01    2     2.804639  0.6970563  2.179830
##   0.01    3     2.747941  0.7113945  2.159689
##   0.01    4     3.065408  0.6636576  2.394201
##   0.01    5     3.239405  0.6349742  2.538847
##   0.01    6     3.618123  0.5746367  2.821578
##   0.01    7     3.935655  0.5223489  3.084171
##   0.01    8     3.831064  0.5389188  3.021469
##   0.01    9     3.650763  0.5720477  2.911138
##   0.01   10     3.838880  0.5328379  3.062723
##   0.10    1     2.505958  0.7513321  1.934889
##   0.10    2     2.790805  0.6999236  2.190519
##   0.10    3     2.749609  0.7158989  2.159751
##   0.10    4     2.825735  0.7007805  2.221362
##   0.10    5     2.965525  0.6799272  2.336499
##   0.10    6     3.354690  0.6172005  2.632584
##   0.10    7     3.335323  0.6168686  2.654745
##   0.10    8     3.712228  0.5392484  2.970709
##   0.10    9     3.432621  0.6031285  2.696727
##   0.10   10     3.321041  0.6249708  2.670665
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were size = 1 and decay = 0.01.
```



Test set

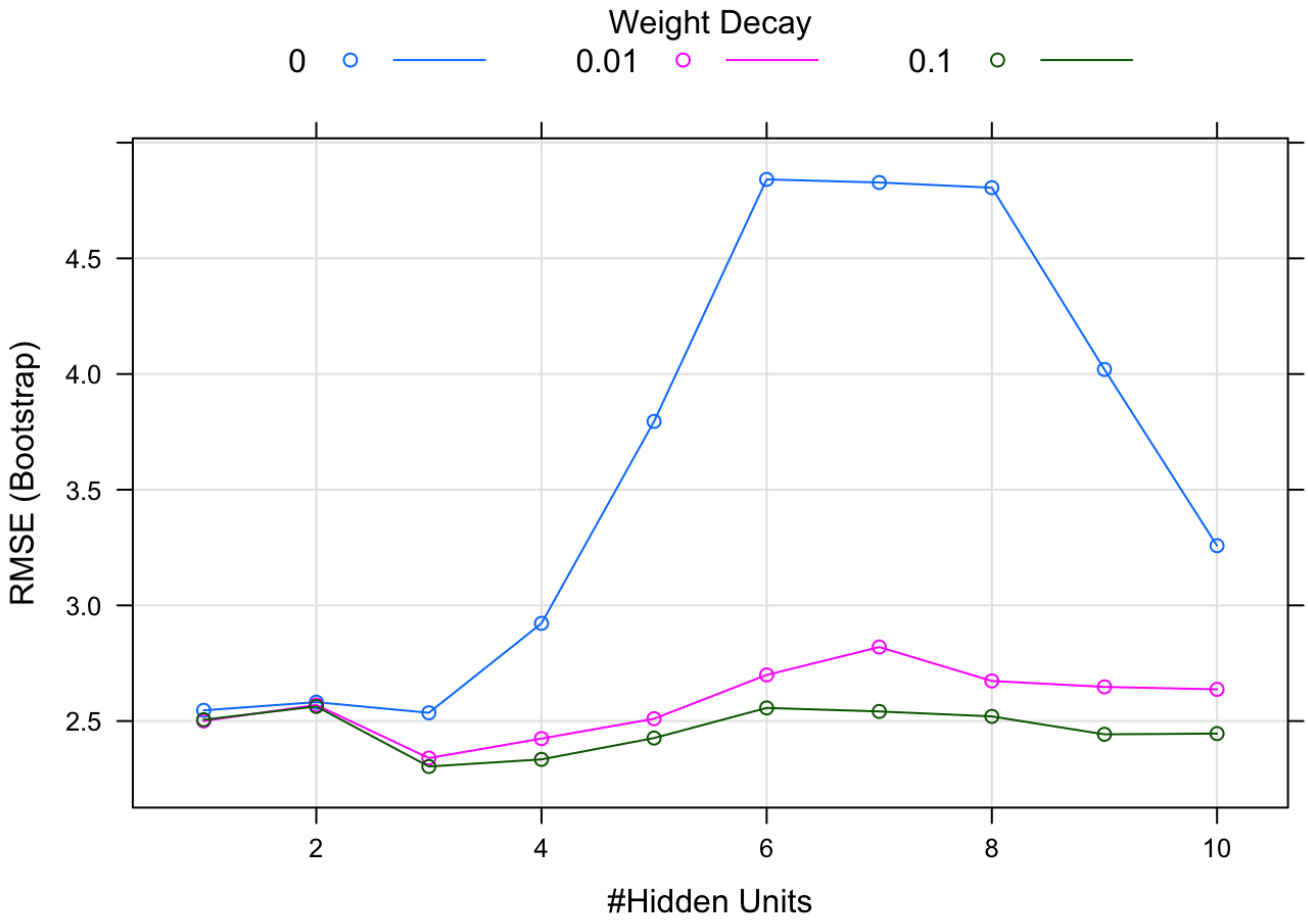
```
##      RMSE  Rsquared       MAE
## 2.6433262 0.7194364 2.0232660
```

From the above plot and the tabulated result, we can clearly see that the best model of neural network model is of size 1 and decay is 0.01. The R^2 value obtained on the test set is 0.7194.

Averaged Neural Network

Lets bulid the model using averaged neural network model using train() in caret package with method as avNNet. I am setting the decay to be 0, 0.001, 0.01, 0.1 and size varying from 1 to 10 with bag as False.

```
## 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    2.546337  0.7439541  1.968793
##   0.00    2    2.581790  0.7400278  2.012679
##   0.00    3    2.535763  0.7515368  1.920367
##   0.00    4    2.922345  0.6958074  2.144579
##   0.00    5    3.795250  0.5939801  2.576917
##   0.00    6    4.841328  0.4831421  3.176902
##   0.00    7    4.827818  0.4718842  3.234973
##   0.00    8    4.805434  0.4642360  3.123138
##   0.00    9    4.019746  0.5945787  2.592198
##   0.00   10    3.257907  0.6450295  2.353564
##   0.01    1    2.500329  0.7521578  1.928185
##   0.01    2    2.569620  0.7404300  2.006646
##   0.01    3    2.339930  0.7852714  1.832702
##   0.01    4    2.424232  0.7737201  1.917992
##   0.01    5    2.510170  0.7522292  1.974175
##   0.01    6    2.699310  0.7275242  2.123788
##   0.01    7    2.819939  0.7043841  2.212574
##   0.01    8    2.673350  0.7251305  2.102813
##   0.01    9    2.647719  0.7301051  2.064986
##   0.01   10    2.637050  0.7280176  2.086017
##   0.10    1    2.505918  0.7513368  1.934844
##   0.10    2    2.563253  0.7429928  1.989915
##   0.10    3    2.303527  0.7889688  1.814756
##   0.10    4    2.334351  0.7835149  1.842384
##   0.10    5    2.426563  0.7702325  1.913187
##   0.10    6    2.556781  0.7469632  2.015201
##   0.10    7    2.541515  0.7480649  2.008697
##   0.10    8    2.520092  0.7524492  1.993615
##   0.10    9    2.442685  0.7664654  1.927898
##   0.10   10    2.445824  0.7672685  1.927857
##
## 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 = 0.1 and bag
##  = FALSE.
```



Test set

```
##      RMSE Rsquared      MAE
## 2.050326 0.833895 1.546593
```

From the above plot and the tabulated result, we can clearly see that the best model of averaged neural network model has weight decay value of 0.1 and size as 3. The R^2 value obtained on the test set is 0.8339.

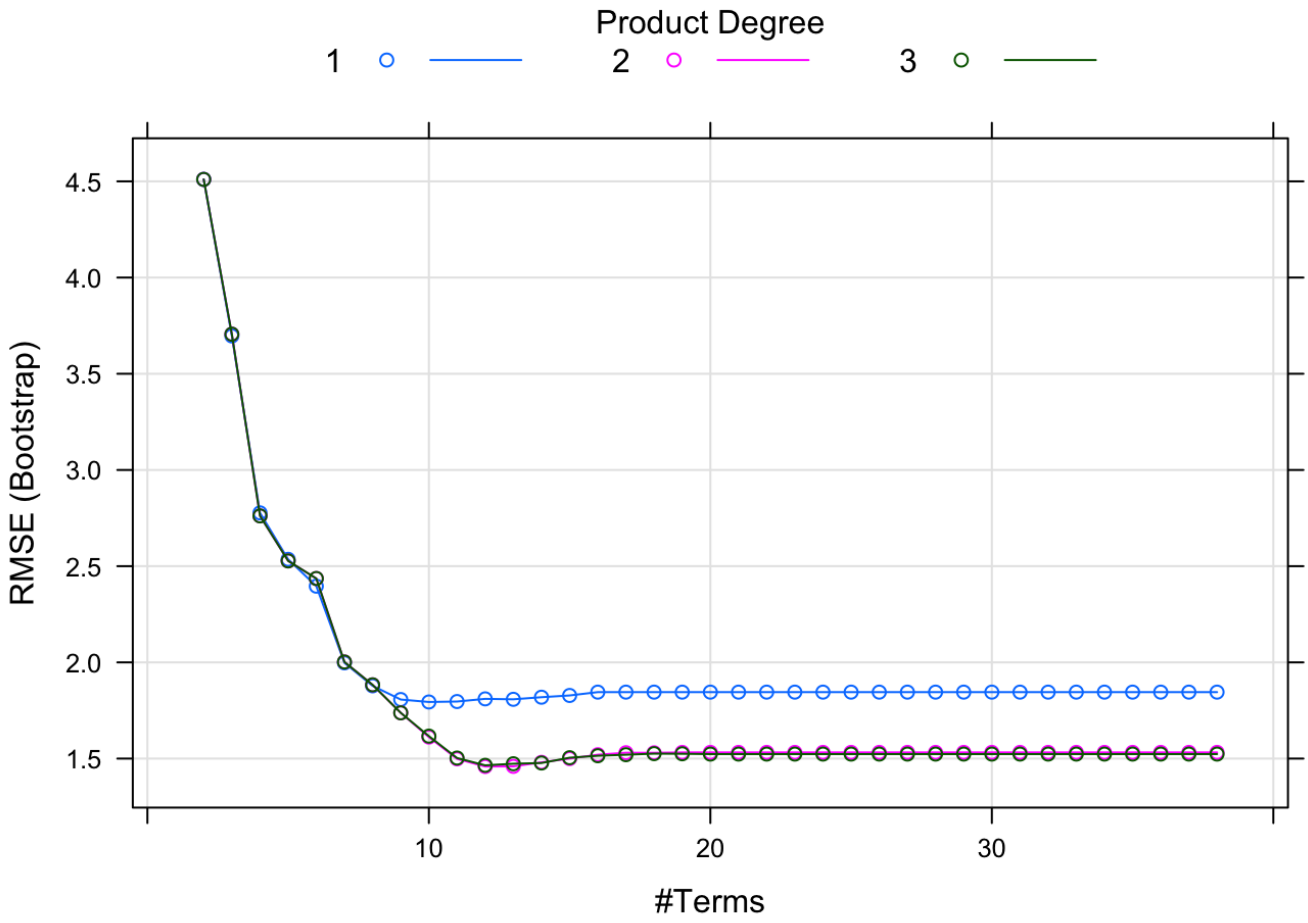
Mars Model with no Preprocessing

Lets bulid the model using mars model using train() in caret package with method as earth and preprocessing. I am setting the degree to be 1, 2, 3 and number of prune varying from 2 to 38.

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      4.510221  0.2017622  3.719596
##   1        3      3.696575  0.4621782  2.992075
##   1        4      2.776976  0.6965425  2.226214
##   1        5      2.535576  0.7442072  2.021430
##   1        6      2.396059  0.7748952  1.910341
```

```
##   1        6      2.590659  0.7746952  1.910541
##   1        7      1.997451  0.8427290  1.588618
##   1        8      1.878071  0.8607824  1.486225
##   1        9      1.806903  0.8709898  1.408215
##   1       10      1.794167  0.8728956  1.387328
##   1       11      1.796483  0.8729617  1.390498
##   1       12      1.810352  0.8714632  1.403777
##   1       13      1.808039  0.8718113  1.395422
##   1       14      1.818921  0.8702901  1.405284
##   1       15      1.828042  0.8690235  1.416676
##   1       16      1.845366  0.8672717  1.429085
##   1       17      1.845366  0.8672717  1.429085
##   1       18      1.845366  0.8672717  1.429085
##   1       19      1.845366  0.8672717  1.429085
##   1       20      1.845366  0.8672717  1.429085
##   1       21      1.845366  0.8672717  1.429085
##   1       22      1.845366  0.8672717  1.429085
##   1       23      1.845366  0.8672717  1.429085
##   1       24      1.845366  0.8672717  1.429085
##   1       25      1.845366  0.8672717  1.429085
##   1       26      1.845366  0.8672717  1.429085
##   1       27      1.845366  0.8672717  1.429085
##   1       28      1.845366  0.8672717  1.429085
##   1       29      1.845366  0.8672717  1.429085
##   1       30      1.845366  0.8672717  1.429085
##   1       31      1.845366  0.8672717  1.429085
##   1       32      1.845366  0.8672717  1.429085
##   1       33      1.845366  0.8672717  1.429085
##   1       34      1.845366  0.8672717  1.429085
##   1       35      1.845366  0.8672717  1.429085
##   1       36      1.845366  0.8672717  1.429085
##   1       37      1.845366  0.8672717  1.429085
##   1       38      1.845366  0.8672717  1.429085
##   2        2      4.510138  0.2017189  3.719805
##   2        3      3.705720  0.4595804  2.997697
##   2        4      2.761341  0.7004236  2.205055
##   2        5      2.527292  0.7476576  2.003029
##   2        6      2.435956  0.7666567  1.918751
##   2        7      2.002104  0.8394419  1.592260
##   2        8      1.883270  0.8591193  1.467498
##   2        9      1.737783  0.8800273  1.352830
##   2       10      1.611688  0.8960040  1.263020
##   2       11      1.498473  0.9099346  1.183164
##   2       12      1.458763  0.9144646  1.162416
##   2       13      1.459539  0.9159286  1.146368
##   2       14      1.480467  0.9140997  1.159628
##   2       15      1.499974  0.9116557  1.173345
##   2       16      1.520221  0.9099219  1.183261
##   2       17      1.530444  0.9085721  1.191702
##   2       18      1.530076  0.9085017  1.186946
##   2       19      1.532100  0.9081181  1.187963
##   2       20      1.532100  0.9081181  1.187963
##   2       21      1.532100  0.9081181  1.187963
##   2       22      1.532100  0.9081181  1.187963
##   2       23      1.532100  0.9081181  1.187963
##   2       24      1.532100  0.9081181  1.187963
##   2       25      1.532100  0.9081181  1.187963
##   2       26      1.532100  0.9081181  1.187963
```

```
##   2       26      1.532100  0.9081181  1.187963
##   2       27      1.532100  0.9081181  1.187963
##   2       28      1.532100  0.9081181  1.187963
##   2       29      1.532100  0.9081181  1.187963
##   2       30      1.532100  0.9081181  1.187963
##   2       31      1.532100  0.9081181  1.187963
##   2       32      1.532100  0.9081181  1.187963
##   2       33      1.532100  0.9081181  1.187963
##   2       34      1.532100  0.9081181  1.187963
##   2       35      1.532100  0.9081181  1.187963
##   2       36      1.532100  0.9081181  1.187963
##   2       37      1.532100  0.9081181  1.187963
##   2       38      1.532100  0.9081181  1.187963
##   3        2      4.510138  0.2017189  3.719805
##   3        3      3.705720  0.4595804  2.997697
##   3        4      2.761341  0.7004236  2.205055
##   3        5      2.527292  0.7476576  2.003029
##   3        6      2.435956  0.7666567  1.918751
##   3        7      2.002104  0.8394419  1.592260
##   3        8      1.883270  0.8591193  1.467498
##   3        9      1.737783  0.8800273  1.352830
##   3       10      1.616709  0.8953253  1.269326
##   3       11      1.502878  0.9095382  1.185268
##   3       12      1.464964  0.9137778  1.164108
##   3       13      1.473203  0.9147496  1.154037
##   3       14      1.477041  0.9139106  1.159154
##   3       15      1.504577  0.9112614  1.177719
##   3       16      1.514711  0.9102665  1.178596
##   3       17      1.520085  0.9091042  1.182633
##   3       18      1.525661  0.9080718  1.187705
##   3       19      1.524936  0.9081347  1.186952
##   3       20      1.523364  0.9082714  1.185232
##   3       21      1.523364  0.9082714  1.185232
##   3       22      1.523364  0.9082714  1.185232
##   3       23      1.523364  0.9082714  1.185232
##   3       24      1.523364  0.9082714  1.185232
##   3       25      1.523364  0.9082714  1.185232
##   3       26      1.523364  0.9082714  1.185232
##   3       27      1.523364  0.9082714  1.185232
##   3       28      1.523364  0.9082714  1.185232
##   3       29      1.523364  0.9082714  1.185232
##   3       30      1.523364  0.9082714  1.185232
##   3       31      1.523364  0.9082714  1.185232
##   3       32      1.523364  0.9082714  1.185232
##   3       33      1.523364  0.9082714  1.185232
##   3       34      1.523364  0.9082714  1.185232
##   3       35      1.523364  0.9082714  1.185232
##   3       36      1.523364  0.9082714  1.185232
##   3       37      1.523364  0.9082714  1.185232
##   3       38      1.523364  0.9082714  1.185232
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were nprune = 12 and degree = 2.
```



Test set

```
##      RMSE  Rsquared       MAE
## 1.2803060 0.9335241 1.0168673
```

From the above plot and the tabulated result, we can clearly see that the best model of mars model has nprune = 12 and degree = 2. The R^2 value obtained on the test set is 0.9335

MARS with spatial sign and correlated predictors removed

Lets bulid the model using mars model using train() in caret package with method as earth and spatial signe. I am setting the degree to be 1, 2, 3 and number of prune varying from 2 to 38. I am checking the correlation between the predictors using .75 cut off and it seems that there is no highly correlated predictors. And upon buliding the model, we get

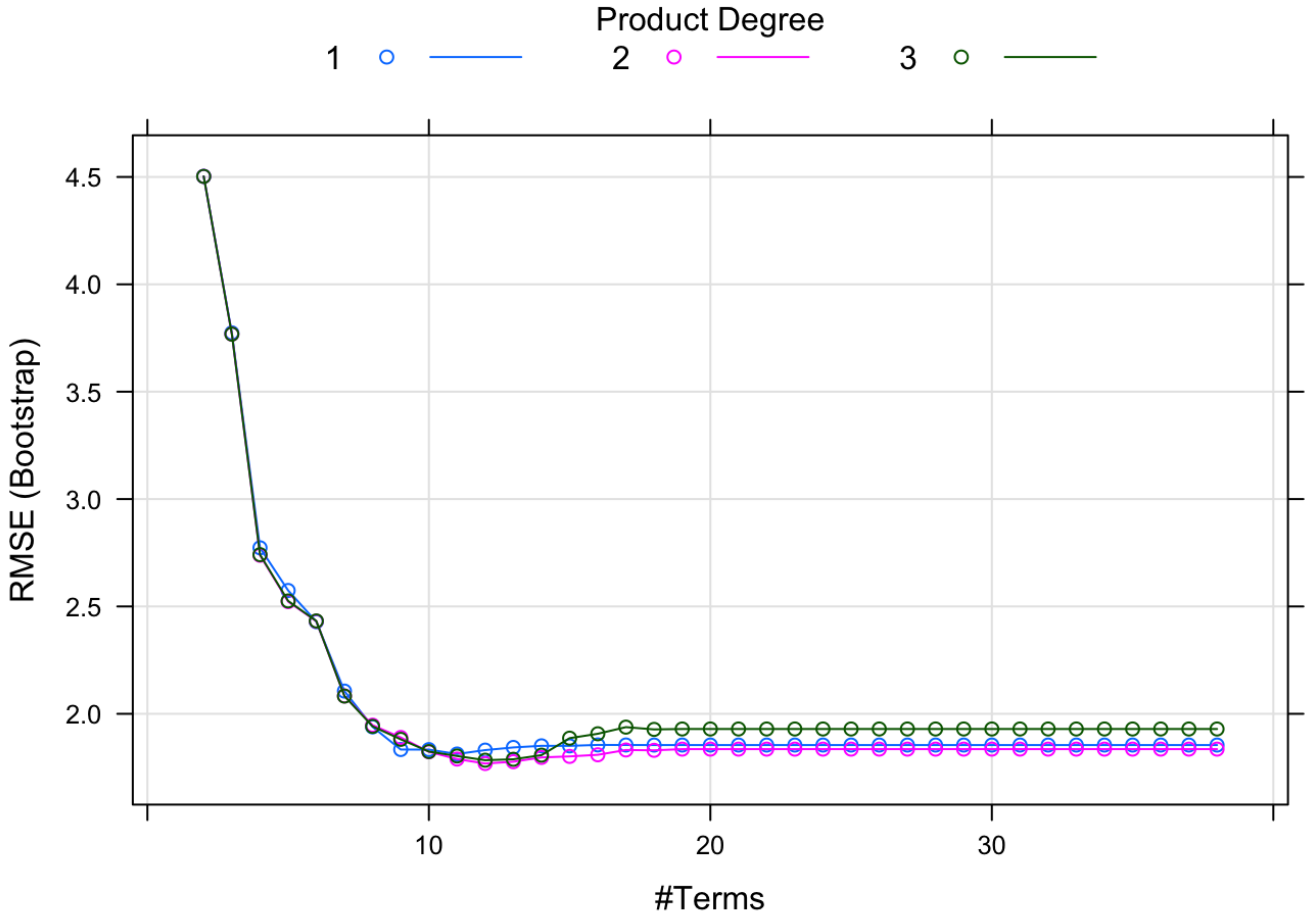
```
## [1] "No of highly correlated of predictors 0"
```

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## Pre-processing: spatial sign transformation (10), centered (10),
##  scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
```

```
##   1        2      4.502657  0.2061144  3.719368
##   1        3      3.773752  0.4401085  3.084723
##   1        4      2.773083  0.6957729  2.229113
##   1        5      2.574324  0.7376381  2.070982
##   1        6      2.428897  0.7660850  1.943801
##   1        7      2.105864  0.8233141  1.687090
##   1        8      1.939120  0.8510747  1.535309
##   1        9      1.833732  0.8684426  1.450762
##   1       10      1.833393  0.8675709  1.444474
##   1       11      1.812792  0.8707369  1.428323
##   1       12      1.831052  0.8677040  1.448415
##   1       13      1.843422  0.8662261  1.456801
##   1       14      1.850948  0.8649641  1.465971
##   1       15      1.850709  0.8648144  1.466267
##   1       16      1.855167  0.8642975  1.466309
##   1       17      1.854787  0.8640049  1.468701
##   1       18      1.854516  0.8639931  1.467581
##   1       19      1.854516  0.8639931  1.467581
##   1       20      1.854516  0.8639931  1.467581
##   1       21      1.854516  0.8639931  1.467581
##   1       22      1.854516  0.8639931  1.467581
##   1       23      1.854516  0.8639931  1.467581
##   1       24      1.854516  0.8639931  1.467581
##   1       25      1.854516  0.8639931  1.467581
##   1       26      1.854516  0.8639931  1.467581
##   1       27      1.854516  0.8639931  1.467581
##   1       28      1.854516  0.8639931  1.467581
##   1       29      1.854516  0.8639931  1.467581
##   1       30      1.854516  0.8639931  1.467581
##   1       31      1.854516  0.8639931  1.467581
##   1       32      1.854516  0.8639931  1.467581
##   1       33      1.854516  0.8639931  1.467581
##   1       34      1.854516  0.8639931  1.467581
##   1       35      1.854516  0.8639931  1.467581
##   1       36      1.854516  0.8639931  1.467581
##   1       37      1.854516  0.8639931  1.467581
##   1       38      1.854516  0.8639931  1.467581
##   2        2      4.502657  0.2061144  3.719368
##   2        3      3.768113  0.4426021  3.082706
##   2        4      2.738270  0.7048160  2.195107
##   2        5      2.522389  0.7504584  2.013110
##   2        6      2.431165  0.7679687  1.933879
##   2        7      2.083178  0.8275236  1.662266
##   2        8      1.947056  0.8498818  1.542497
##   2        9      1.889897  0.8569999  1.495610
##   2       10      1.823159  0.8664777  1.440644
##   2       11      1.789391  0.8717263  1.414478
##   2       12      1.768563  0.8750622  1.392564
##   2       13      1.776977  0.8741675  1.400608
##   2       14      1.796977  0.8707876  1.419468
##   2       15      1.801702  0.8707804  1.412562
##   2       16      1.809158  0.8698855  1.417217
##   2       17      1.831595  0.8671231  1.431107
##   2       18      1.830067  0.8667817  1.426796
##   2       19      1.835542  0.8661158  1.430227
##   2       20      1.835542  0.8661158  1.430227
##   2       21      1.835542  0.8661158  1.430227
```

```
##   2        21      1.835542  0.8661158  1.430227
##   2        22      1.835542  0.8661158  1.430227
##   2        23      1.835542  0.8661158  1.430227
##   2        24      1.835542  0.8661158  1.430227
##   2        25      1.835542  0.8661158  1.430227
##   2        26      1.835542  0.8661158  1.430227
##   2        27      1.835542  0.8661158  1.430227
##   2        28      1.835542  0.8661158  1.430227
##   2        29      1.835542  0.8661158  1.430227
##   2        30      1.835542  0.8661158  1.430227
##   2        31      1.835542  0.8661158  1.430227
##   2        32      1.835542  0.8661158  1.430227
##   2        33      1.835542  0.8661158  1.430227
##   2        34      1.835542  0.8661158  1.430227
##   2        35      1.835542  0.8661158  1.430227
##   2        36      1.835542  0.8661158  1.430227
##   2        37      1.835542  0.8661158  1.430227
##   2        38      1.835542  0.8661158  1.430227
##   3         2      4.502657  0.2061144  3.719368
##   3         3      3.768113  0.4426021  3.082706
##   3         4      2.741188  0.7041199  2.198031
##   3         5      2.525605  0.7498857  2.012894
##   3         6      2.432509  0.7678354  1.932557
##   3         7      2.082609  0.8276537  1.660925
##   3         8      1.941546  0.8510424  1.539333
##   3         9      1.881068  0.8593138  1.486842
##   3        10      1.824229  0.8672745  1.439056
##   3        11      1.803584  0.8704600  1.424008
##   3        12      1.783990  0.8733145  1.405381
##   3        13      1.788325  0.8729412  1.406515
##   3        14      1.807585  0.8690873  1.425125
##   3        15      1.886553  0.8594084  1.433359
##   3        16      1.907032  0.8576607  1.441314
##   3        17      1.937961  0.8536091  1.458669
##   3        18      1.927383  0.8550265  1.449000
##   3        19      1.929311  0.8547459  1.449531
##   3        20      1.929311  0.8547459  1.449531
##   3        21      1.929311  0.8547459  1.449531
##   3        22      1.929311  0.8547459  1.449531
##   3        23      1.929311  0.8547459  1.449531
##   3        24      1.929311  0.8547459  1.449531
##   3        25      1.929311  0.8547459  1.449531
##   3        26      1.929311  0.8547459  1.449531
##   3        27      1.929311  0.8547459  1.449531
##   3        28      1.929311  0.8547459  1.449531
##   3        29      1.929311  0.8547459  1.449531
##   3        30      1.929311  0.8547459  1.449531
##   3        31      1.929311  0.8547459  1.449531
##   3        32      1.929311  0.8547459  1.449531
##   3        33      1.929311  0.8547459  1.449531
##   3        34      1.929311  0.8547459  1.449531
##   3        35      1.929311  0.8547459  1.449531
##   3        36      1.929311  0.8547459  1.449531
##   3        37      1.929311  0.8547459  1.449531
##   3        38      1.929311  0.8547459  1.449531
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were n.trees = 10, shrinkage = 0
```

```
## The final values used for the model were nprune = 12 and degree = 2.
```



Test set

```
##      RMSE  Rsquared       MAE
## 1.6543157 0.8890979 1.3021739
```

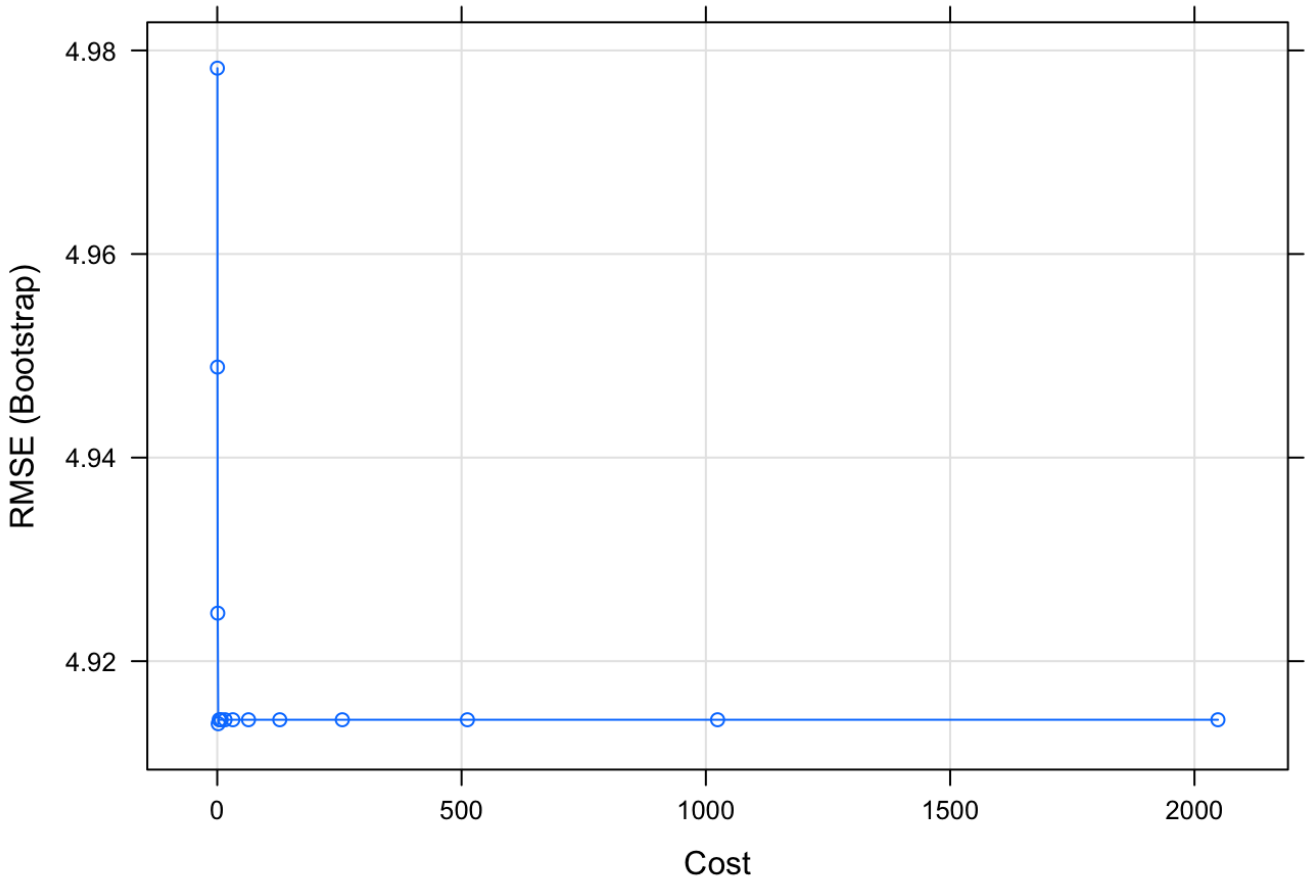
From the above plot and the tabulated result, we can clearly see that the best model of mars model has nprune = 12 and degree = 2. The R^2 value obtained on the test set is 0.8891.

Thus by comparing to the above MARS model with no preprocessing R square value of 0.9335 this is quite low. Thus MARS model with no spatial transfrom is performing better on comparsion.

Support Vector Machine

Lets bulid the model using support vector machine using train() in caret package with svmRadial method which uses radial basis function. For svm, I set the tune length to be 14 as it tuneLength argument will use the default grid search of 20 cost values between 2^-2, 2^-1, . . . , 2^11. sigma is estimated analytically by default

```
## 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  4.978256  0.1218695  4.085782
##      0.50  4.948892  0.1339104  4.059287
##      1.00  4.924717  0.1424149  4.040261
##      2.00  4.913855  0.1525388  4.031108
##      4.00  4.914246  0.1529141  4.030790
##      8.00  4.914246  0.1529141  4.030790
##     16.00  4.914246  0.1529141  4.030790
##     32.00  4.914246  0.1529141  4.030790
##     64.00  4.914246  0.1529141  4.030790
##    128.00  4.914246  0.1529141  4.030790
##    256.00  4.914246  0.1529141  4.030790
##    512.00  4.914246  0.1529141  4.030790
##   1024.00  4.914246  0.1529141  4.030790
##   2048.00  4.914246  0.1529141  4.030790
##
## Tuning parameter 'sigma' was held constant at a value of 0.06444911
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were sigma = 0.06444911 and C = 2.
```



Test set

```
##      RMSE  Rsquared       MAE
## 4.8632139 0.2051648 3.9592115
```

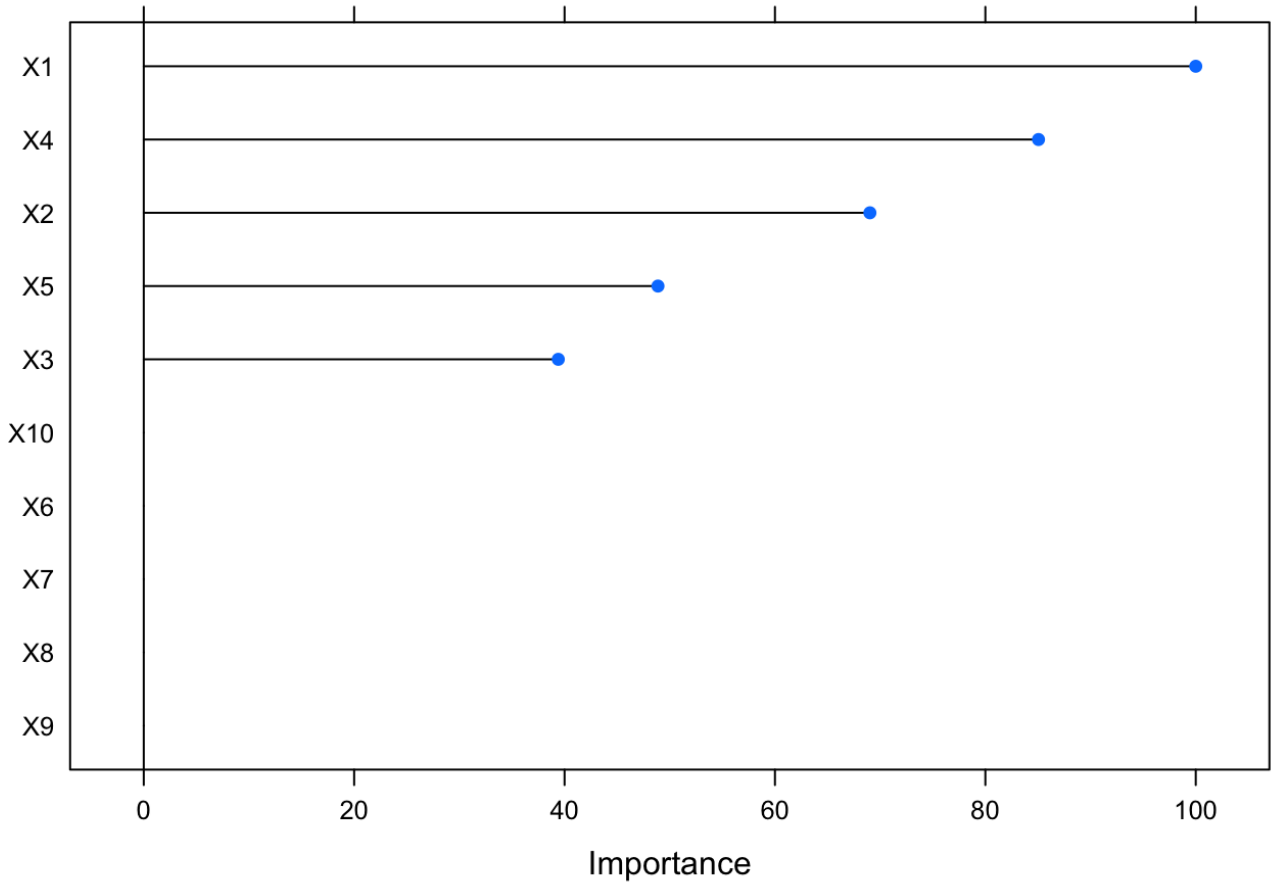
From the above plot and the tabulated result, we can clearly see that the final values used for the model were sigma = 0.06444911 and C = 2. The R^2 value obtained on the test set is 0.2052.

Thus from buliding up all the above models, I have the following table.

Model	Parameter	Training RMSE	Training R Squared	Testing RMSE	Testing R Squared
KNN	k as 15	3.2916	0.6124	3.1751	0.6786
Neural Network	size as 1 & decay as 0.01	2.5028	0.7526	2.6433	0.7194
Averaged Neural Network	size as 3 & decay as 0.1	2.3035	0.789	2.050	0.8339
MARS with no Preprocessing	nprune as 12 & degree as 2	1.4588	0.9145	1.2803	0.9335
MARS with spatial sign	nprune as 12 &	1.7686	0.8751	1.6543	0.8891

Model	Parameter	Training RMSE	Training R Squared	Testing RMSE	Testing R Squared
and correlated predictors removed	degree as 2				
Support Vector Machine	C as 2 & sigma as 0.06445	4.9139	0.1526	4.8632	0.2052

So from the table we can see that MARS model with no preprocessing gives best prediction as it has highest R^2 value 0.9335 on the test set. To answer the question regarding the predictor's importance on the MARS model I used varImp() and plotted below plot.



Now from the above plot we can clearly see that MARS model select only the informative predcitors that is x1 - x5 (in the order x2, x4, x1, x5, x3), as x6 to x10 has values as 0.

Question 3

For this question we are going to work with the Tecator dataset and bulid various non linear regression models over it. Tecator dataset is a combination of absorp contains the 100 absorbance values for the 215 samples and endpoints contains the percent of moisture, fat, and protein in columns 1–3, respectively. Lets first load the data here.

```
##        V1      V2      V3      V4      V5
## 1 2.61776 2.61814 2.61859 2.61912 2.61981
## 2 2.83454 2.83871 2.84283 2.84705 2.85138
## 3 2.58284 2.58458 2.58629 2.58808 2.58996
## 4 2.82286 2.82460 2.82630 2.82814 2.83001
## 5 2.78813 2.78989 2.79167 2.79350 2.79538
## 6 3.00993 3.01540 3.02086 3.02634 3.03190
```

```
##   mositure  fat protein
## 1     60.5 22.5    16.7
## 2     46.0 40.1    13.5
## 3     71.0  8.4    20.5
## 4     72.8  5.9    20.7
## 5     58.3 25.5    15.5
## 6     44.0 42.7    13.7
```

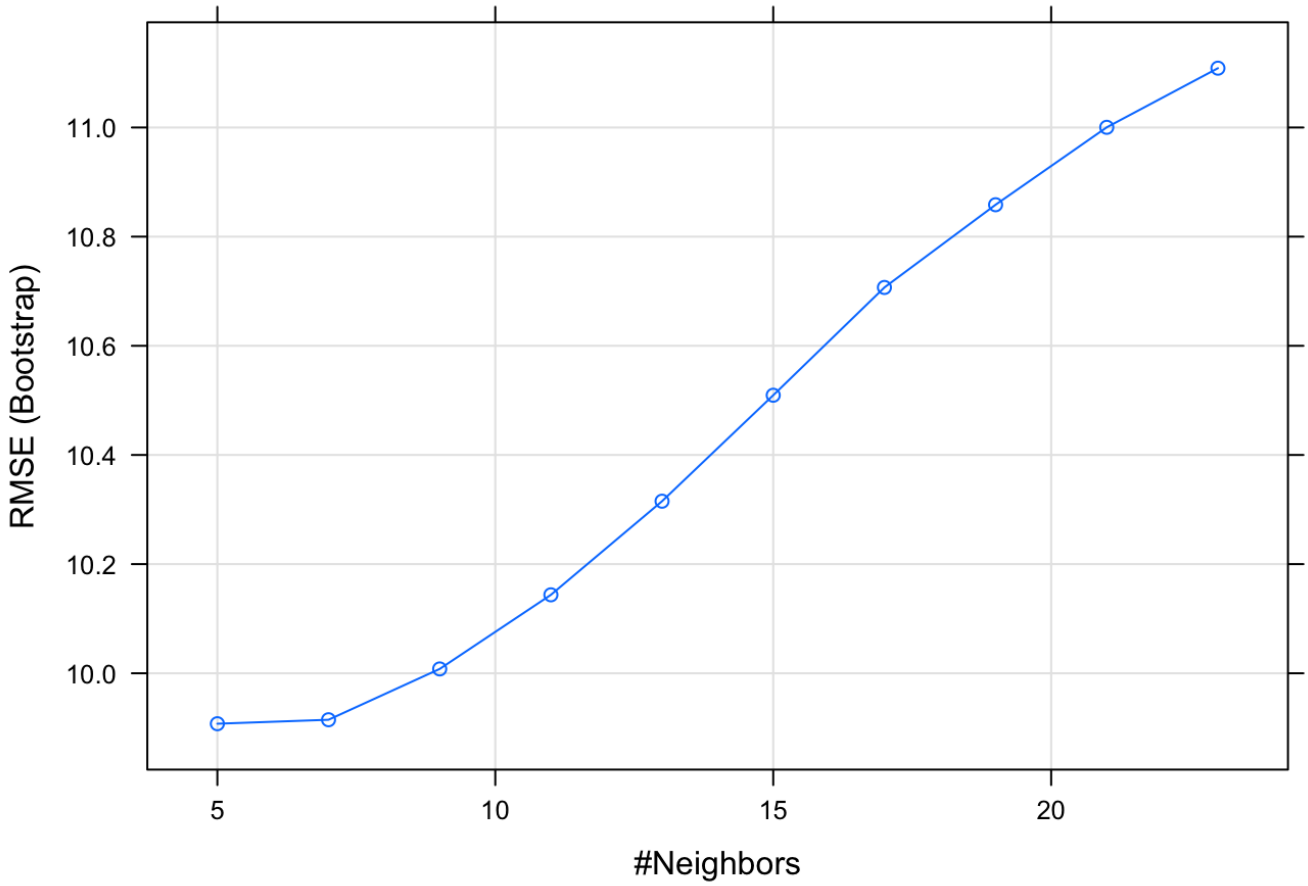
Thus we have above the data loaded properly. Lets get started with buliding the models.

Before we generate the model we need to split the data into training and testing samples. Given the sample size, we will retain the 80% of the samples to the training set and 20% of the sample in the testing set. The train set will be used to tune the models by splitting that into 10 fold for cross validation in order to have better model performance. For spliting the train set we will use Leave group out cross validation with 5 folds.

KNN

Lets first fit the data with a KNN model. For this I have used train() in the caret package with knn as method. Below we have complete information about the KNN model.

```
## k-Nearest Neighbors
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 174, 174, 174, 174, 174, 174, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    5   9.907561  0.4487340  7.395216
##    7   9.914853  0.4416734  7.620026
##    9  10.007944  0.4310662  7.796599
##   11  10.143735  0.4216959  8.032671
##   13  10.315272  0.4055667  8.253291
##   15  10.509426  0.3858847  8.452882
##   17  10.706905  0.3648644  8.630340
##   19  10.858457  0.3491532  8.756594
##   21  11.000344  0.3349530  8.893459
##   23  11.108659  0.3231675  8.991943
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was k = 5.
```



Test set

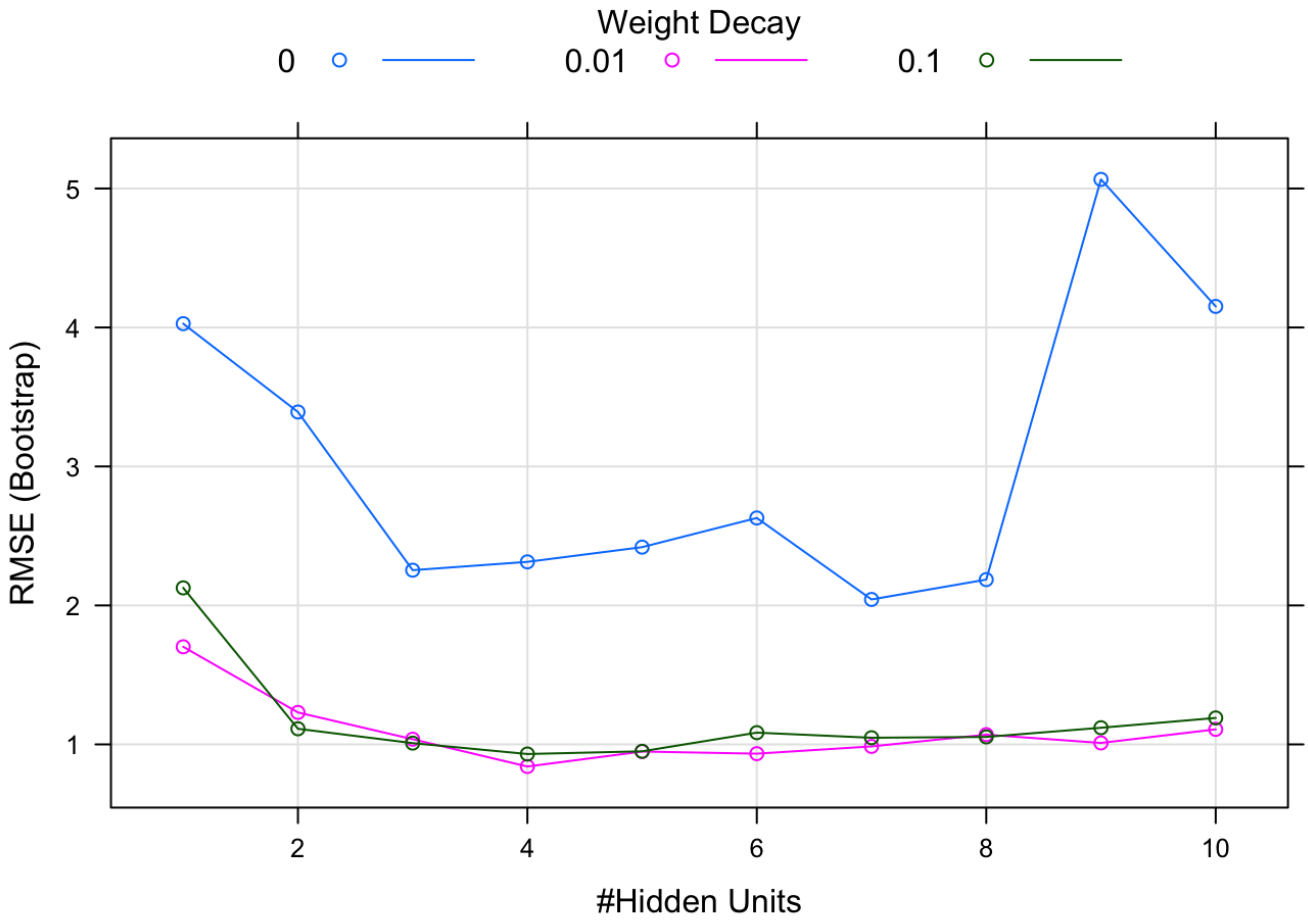
```
##      RMSE  Rsquared       MAE
## 8.3321816 0.4499784 6.3130081
```

From the above plot and the tabulated result, we can clearly see that the best model has the k value of 5. That is the best R^2 is obtain when we are considering 5 nearest neighbours. The R^2 value obtained on the test set is 0.45.

Neural Network without PCA

Lets bulid the model using neural network model using train() in caret package with method as nnet. Here we are buliding it without PCA. I am setting the decay to be 0, 0.001, 0.01, 0.1 and size varying from 1 to 10.

```
## Neural Network
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 174, 174, 174, 174, 174, 174, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE       Rsquared   MAE
##   0.00    1    4.0274029  0.8707140  2.7064810
##   0.00    2    3.3918999  0.9149002  2.1840546
##   0.00    3    2.2536621  0.9675529  1.3435654
##   0.00    4    2.3134991  0.9622970  1.3814594
##   0.00    5    2.4193988  0.9565166  1.2195736
##   0.00    6    2.6292965  0.9477805  1.2416843
##   0.00    7    2.0429444  0.9708908  1.1736787
##   0.00    8    2.1858990  0.9686163  1.2100587
##   0.00    9    5.0655687  0.8340335  2.2747451
##   0.00   10    4.1515098  0.8992701  1.7147063
##   0.01    1    1.7023423  0.9835957  1.2973041
##   0.01    2    1.2300328  0.9908059  0.8871585
##   0.01    3    1.0373540  0.9926728  0.7341144
##   0.01    4    0.8409236  0.9953549  0.5964121
##   0.01    5    0.9487652  0.9932432  0.6528657
##   0.01    6    0.9333352  0.9942958  0.6419682
##   0.01    7    0.9855168  0.9932686  0.7005875
##   0.01    8    1.0702981  0.9933602  0.7114297
##   0.01    9    1.0104963  0.9941262  0.7101267
##   0.01   10    1.1082070  0.9927349  0.7680613
##   0.10    1    2.1262044  0.9743860  1.6912023
##   0.10    2    1.1124714  0.9917564  0.8465395
##   0.10    3    1.0092120  0.9929286  0.7631179
##   0.10    4    0.9308673  0.9945179  0.7067223
##   0.10    5    0.9502652  0.9945364  0.7031983
##   0.10    6    1.0846410  0.9914797  0.8022170
##   0.10    7    1.0479175  0.9935813  0.7661535
##   0.10    8    1.0539272  0.9935898  0.7751945
##   0.10    9    1.1198449  0.9924372  0.7989222
##   0.10   10    1.1904466  0.9916675  0.8556912
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were size = 4 and decay = 0.01.
```



Test set

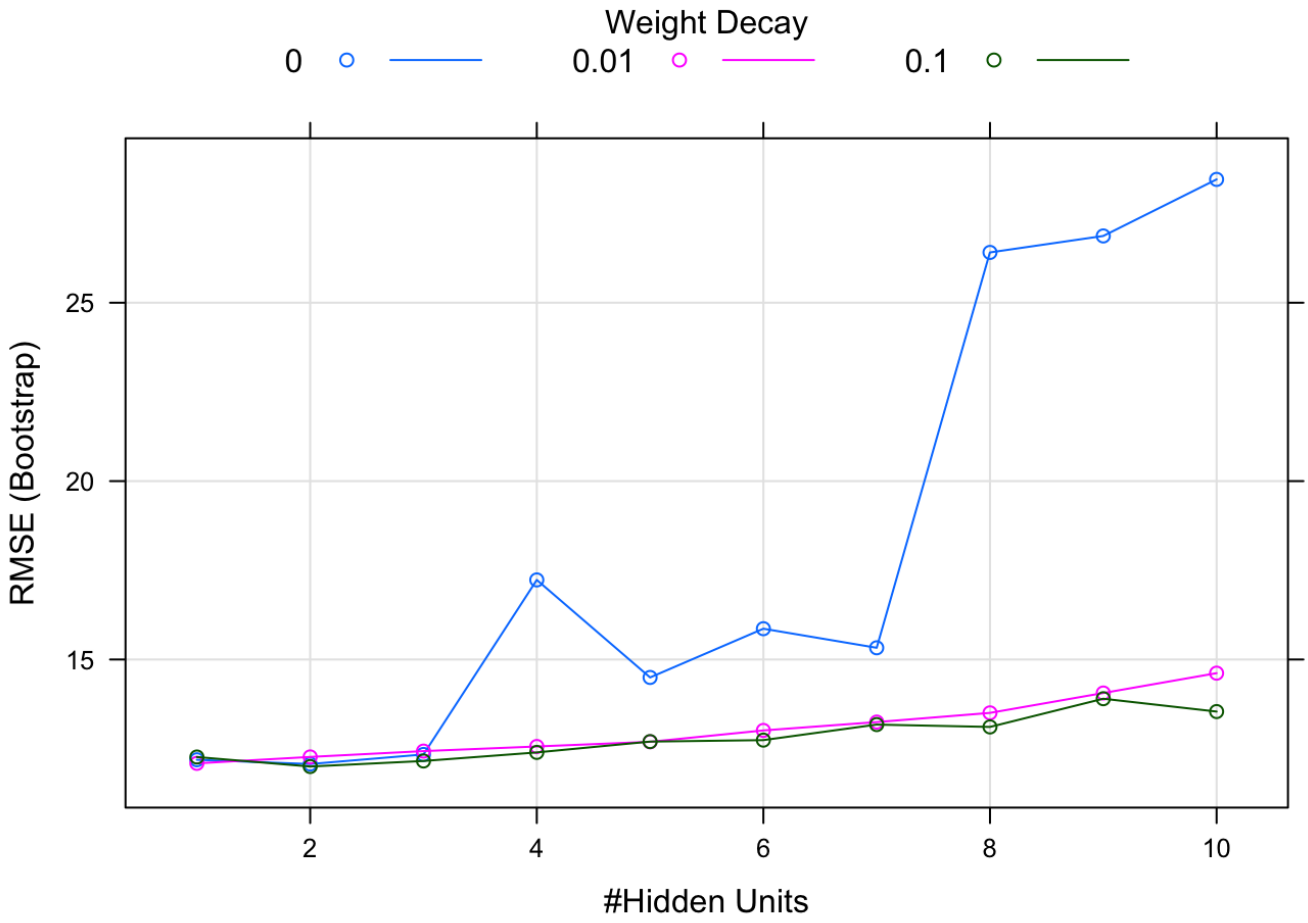
```
##      RMSE  Rsquared       MAE
## 0.4271682 0.9987065 0.3236428
```

From the above plot and the tabulated result, we can clearly see that the best model of neural network model is of size 4 and decay is 0.01. The R^2 value obtained on the test set is 0.9987.

Neural Network with PCA

Let us bulid the model using PCA in the preprocessing argument of the train(). I am setting the decay to be 0, 0.001, 0.01, 0.1 and size varying from 1 to 10.

```
## Neural Network
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100), principal component
##  signal extraction (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 174, 174, 174, 174, 174, 174, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE      Rsquared    MAE
##   0.00    1    12.18957  0.17306810   9.752068
##   0.00    2    12.07030  0.19394648   9.456260
##   0.00    3    12.33844  0.18108846   9.618613
##   0.00    4    17.23042  0.16965496  10.496844
##   0.00    5    14.49690  0.16442927  10.477867
##   0.00    6    15.86325  0.13896938  10.776789
##   0.00    7    15.32804  0.12527628  10.873554
##   0.00    8    26.41007  0.10204171  13.027818
##   0.00    9    26.87176  0.09186633  13.184149
##   0.00   10    28.45709  0.10217491  13.527934
##   0.01    1    12.08913  0.19165966   9.649027
##   0.01    2    12.26940  0.17826014   9.644300
##   0.01    3    12.43257  0.17869613   9.614212
##   0.01    4    12.56188  0.17500362   9.737005
##   0.01    5    12.69277  0.18150708   9.860975
##   0.01    6    13.01163  0.16285676   9.946475
##   0.01    7    13.24589  0.15658427  10.143004
##   0.01    8    13.50368  0.15374787  10.252936
##   0.01    9    14.06039  0.15797106  10.432378
##   0.01   10    14.61695  0.13208085  10.712957
##   0.10    1    12.26677  0.16607621   9.845121
##   0.10    2    12.00036  0.20539528   9.405782
##   0.10    3    12.15547  0.20013157   9.493548
##   0.10    4    12.39563  0.18840267   9.605633
##   0.10    5    12.69874  0.16371904   9.838733
##   0.10    6    12.74032  0.17380835   9.782252
##   0.10    7    13.17575  0.15112692  10.055735
##   0.10    8    13.10992  0.18480536   9.970003
##   0.10    9    13.90083  0.14257811  10.364962
##   0.10   10    13.54059  0.15092712  10.238430
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were size = 2 and decay = 0.1.
```



Test set

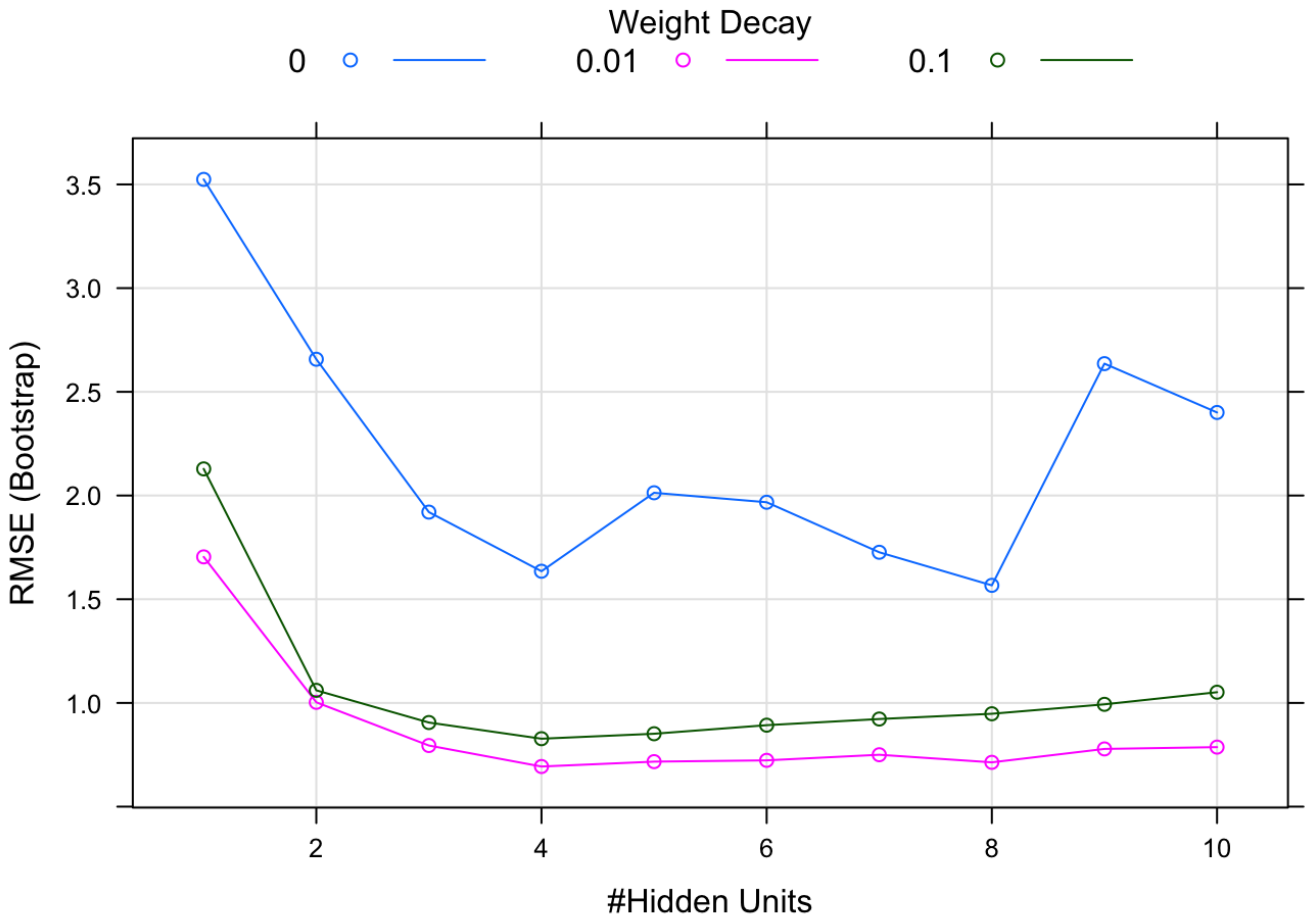
```
##      RMSE  Rsquared       MAE
## 9.5492911 0.2661236 7.6991560
```

From the above plot and the tabulated result, we can clearly see that the best model of neural network model is of size 2 and decay is 0.1. The R^2 value obtained on the test set is 0.2661. It is very low compared to neural network without pcs. Thus PCA is not helping much.

Averaged Neural Network

Lets bulid the model using averaged neural network model using train() in caret package with method as avNNet. I am setting the decay to be 0, 0.001, 0.01, 0.1 and size varying from 1 to 10 with bag as False.

```
## Model Averaged Neural Network
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 174, 174, 174, 174, 174, 174, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE       Rsquared   MAE
##   0.00    1    3.5242925  0.9328497  2.5045521
##   0.00    2    2.6571279  0.9583170  1.7362607
##   0.00    3    1.9201062  0.9738883  1.1502046
##   0.00    4    1.6349793  0.9841445  0.9848295
##   0.00    5    2.0133071  0.9703377  1.0801209
##   0.00    6    1.9676251  0.9688311  0.9854313
##   0.00    7    1.7264409  0.9798391  0.9438506
##   0.00    8    1.5669336  0.9851564  0.9096512
##   0.00    9    2.6359269  0.9427061  1.2948829
##   0.00   10    2.4005105  0.9571086  1.1634948
##   0.01    1    1.7042312  0.9835651  1.2983298
##   0.01    2    1.0029831  0.9942247  0.7364399
##   0.01    3    0.7946900  0.9963027  0.5879969
##   0.01    4    0.6934589  0.9972539  0.5039293
##   0.01    5    0.7170243  0.9970049  0.5162669
##   0.01    6    0.7232355  0.9970309  0.5259775
##   0.01    7    0.7502219  0.9966700  0.5417188
##   0.01    8    0.7136120  0.9970212  0.5240492
##   0.01    9    0.7782966  0.9964293  0.5604093
##   0.01   10    0.7869717  0.9964242  0.5787073
##   0.10    1    2.1284359  0.9743401  1.6929244
##   0.10    2    1.0607973  0.9935692  0.8107018
##   0.10    3    0.9055211  0.9954107  0.6909658
##   0.10    4    0.8275367  0.9961154  0.6323710
##   0.10    5    0.8514127  0.9957906  0.6385393
##   0.10    6    0.8927609  0.9953755  0.6765584
##   0.10    7    0.9224391  0.9950162  0.6956041
##   0.10    8    0.9477885  0.9946943  0.7065720
##   0.10    9    0.9932834  0.9941829  0.7347208
##   0.10   10    1.0517201  0.9934280  0.7704496
##
## 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 = 0.01 and bag
##  = FALSE.
```



Test set

```
##      RMSE  Rsquared       MAE
## 0.3869129 0.9988571 0.3071895
```

From the above plot and the tabulated result, we can clearly see that the best model of averaged neural network model has weight decay value of 0.1 and size as 4. The R^2 value obtained on the test set is 0.9989.

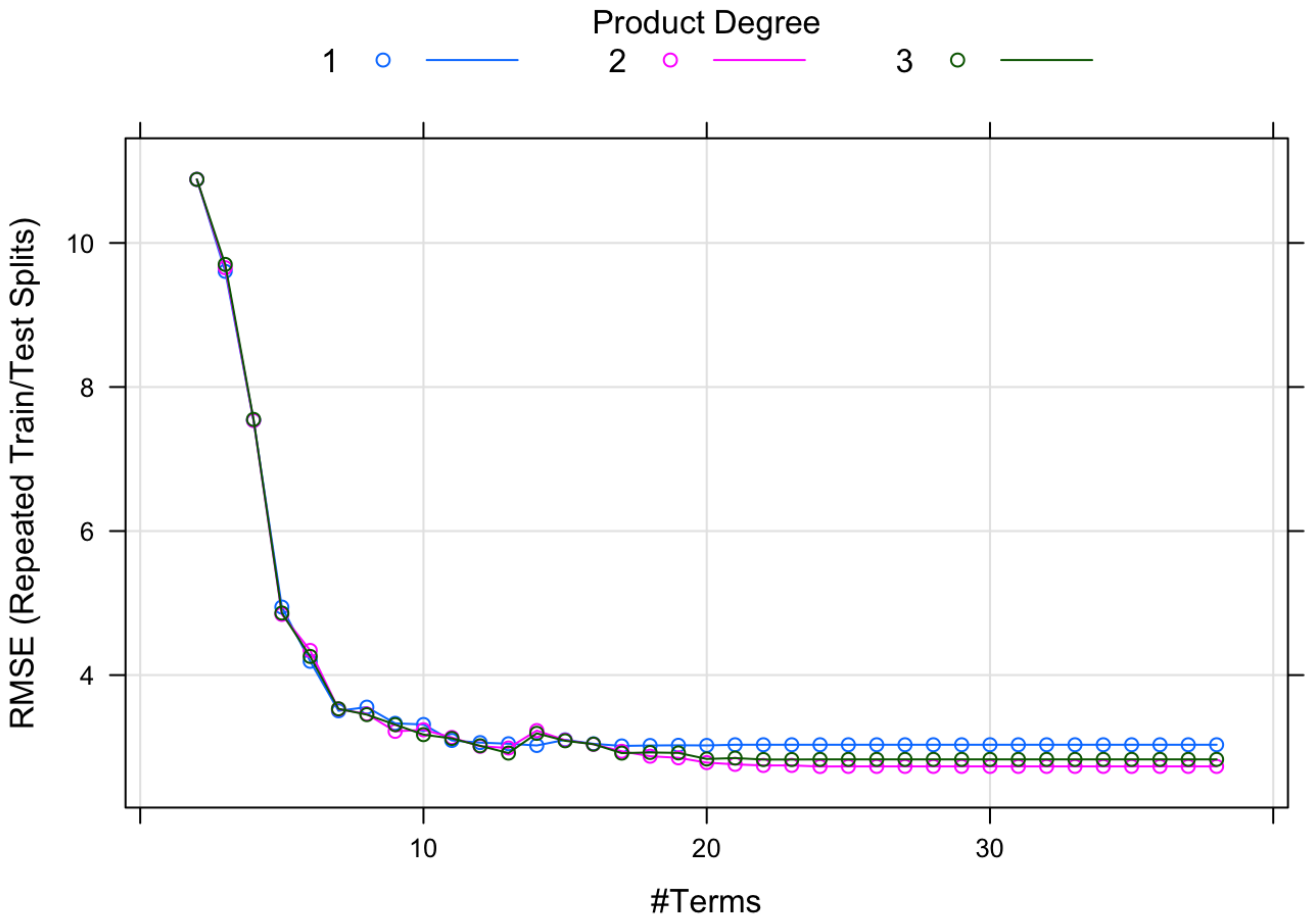
Mars Model with no preprocessing

Lets bulid the model using mars model with train() in caret package with method as earth. I am setting the degree to be 1, 2, 3 and number of prune varying from 2 to 38.

```
## Multivariate Adaptive Regression Spline
##
## 174 samples
## 100 predictors
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 131, 131, 131, 131, 131, 131, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE       Rsquared   MAE
##   1        2      10.882638  0.3213137  8.804301
##   1        3       9.605814  0.4753542  7.440073
##   1        4       7.538237  0.6794047  5.635368
##   1        5       4.944785  0.8646814  3.568613
##   1        6       4.191577  0.9032519  3.190915
```

```
##   1         6       4.191577  0.9032519  3.190915
##   1         7       3.504226  0.9339337  2.741468
##   1         8       3.555427  0.9307751  2.721168
##   1         9       3.330244  0.9399755  2.535762
##   1        10       3.314323  0.9398287  2.464012
##   1        11       3.092622  0.9469798  2.342433
##   1        12       3.062935  0.9482436  2.334861
##   1        13       3.046961  0.9490443  2.295845
##   1        14       3.022933  0.9491133  2.290815
##   1        15       3.101423  0.9462621  2.312860
##   1        16       3.046201  0.9482041  2.266685
##   1        17       3.016728  0.9491976  2.260351
##   1        18       3.022724  0.9491082  2.256020
##   1        19       3.025599  0.9491135  2.256206
##   1        20       3.023716  0.9491272  2.253802
##   1        21       3.033046  0.9488485  2.256340
##   1        22       3.033046  0.9488485  2.256340
##   1        23       3.033046  0.9488485  2.256340
##   1        24       3.033046  0.9488485  2.256340
##   1        25       3.033046  0.9488485  2.256340
##   1        26       3.033046  0.9488485  2.256340
##   1        27       3.033046  0.9488485  2.256340
##   1        28       3.033046  0.9488485  2.256340
##   1        29       3.033046  0.9488485  2.256340
##   1        30       3.033046  0.9488485  2.256340
##   1        31       3.033046  0.9488485  2.256340
##   1        32       3.033046  0.9488485  2.256340
##   1        33       3.033046  0.9488485  2.256340
##   1        34       3.033046  0.9488485  2.256340
##   1        35       3.033046  0.9488485  2.256340
##   1        36       3.033046  0.9488485  2.256340
##   1        37       3.033046  0.9488485  2.256340
##   1        38       3.033046  0.9488485  2.256340
##   2         2      10.882638  0.3213137  8.804301
##   2         3       9.655935  0.4691248  7.518094
##   2         4       7.535579  0.6790415  5.653381
##   2         5       4.843011  0.8715593  3.508081
##   2         6       4.342596  0.8984904  3.265936
##   2         7       3.528121  0.9328108  2.742578
##   2         8       3.460461  0.9364515  2.635235
##   2         9       3.218843  0.9458286  2.433121
##   2        10       3.241067  0.9439320  2.410856
##   2        11       3.134069  0.9446240  2.319778
##   2        12       3.011658  0.9492689  2.215820
##   2        13       2.987953  0.9492075  2.171037
##   2        14       3.229807  0.9443476  2.190484
##   2        15       3.096594  0.9494481  2.106684
##   2        16       3.039415  0.9515502  2.046034
##   2        17       2.942303  0.9545697  1.996892
##   2        18       2.873597  0.9564264  1.949334
##   2        19       2.851413  0.9571525  1.938927
##   2        20       2.785202  0.9586330  1.898244
##   2        21       2.760601  0.9592890  1.883520
##   2        22       2.745493  0.9597077  1.875384
##   2        23       2.746077  0.9597077  1.873338
##   2        24       2.730874  0.9600733  1.859675
##   2        25       2.732238  0.9600345  1.858422
##   2        26       2.730930  0.9600345  1.858422
```

```
##   2       26       2.732238  0.9600345  1.858422
##   2       27       2.732238  0.9600345  1.858422
##   2       28       2.732238  0.9600345  1.858422
##   2       29       2.732238  0.9600345  1.858422
##   2       30       2.732238  0.9600345  1.858422
##   2       31       2.732238  0.9600345  1.858422
##   2       32       2.732238  0.9600345  1.858422
##   2       33       2.732238  0.9600345  1.858422
##   2       34       2.732238  0.9600345  1.858422
##   2       35       2.732238  0.9600345  1.858422
##   2       36       2.732238  0.9600345  1.858422
##   2       37       2.732238  0.9600345  1.858422
##   2       38       2.732238  0.9600345  1.858422
##   3        2      10.882638  0.3213137  8.804301
##   3        3       9.702378  0.4635631  7.573560
##   3        4       7.550224  0.6760330  5.661790
##   3        5       4.861675  0.8703161  3.510451
##   3        6       4.259329  0.9014792  3.250413
##   3        7       3.531824  0.9327056  2.751938
##   3        8       3.451401  0.9363189  2.639899
##   3        9       3.312371  0.9414596  2.474992
##   3       10       3.173599  0.9457034  2.337047
##   3       11       3.120494  0.9444591  2.288823
##   3       12       3.018431  0.9486736  2.205814
##   3       13       2.917149  0.9515388  2.167526
##   3       14       3.191303  0.9460618  2.211166
##   3       15       3.088076  0.9496868  2.123090
##   3       16       3.041126  0.9511874  2.054220
##   3       17       2.916878  0.9552198  1.995997
##   3       18       2.925824  0.9549296  1.999696
##   3       19       2.919763  0.9549975  1.976350
##   3       20       2.835689  0.9571031  1.925702
##   3       21       2.847999  0.9567770  1.917946
##   3       22       2.827825  0.9573626  1.910699
##   3       23       2.827825  0.9573626  1.910699
##   3       24       2.829733  0.9573196  1.909537
##   3       25       2.829733  0.9573196  1.909537
##   3       26       2.829733  0.9573196  1.909537
##   3       27       2.829733  0.9573196  1.909537
##   3       28       2.829733  0.9573196  1.909537
##   3       29       2.829733  0.9573196  1.909537
##   3       30       2.829733  0.9573196  1.909537
##   3       31       2.829733  0.9573196  1.909537
##   3       32       2.829733  0.9573196  1.909537
##   3       33       2.829733  0.9573196  1.909537
##   3       34       2.829733  0.9573196  1.909537
##   3       35       2.829733  0.9573196  1.909537
##   3       36       2.829733  0.9573196  1.909537
##   3       37       2.829733  0.9573196  1.909537
##   3       38       2.829733  0.9573196  1.909537
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were nprune = 24 and degree = 2.
```



Test set

```
##      RMSE  Rsquared       MAE
## 1.3920623 0.9849597 1.1936344
```

From the above plot and the tabulated result, we can clearly see that the best model of mars model has nprune = 24 and degree = 2. The R^2 value obtained on the test set is 0.985.

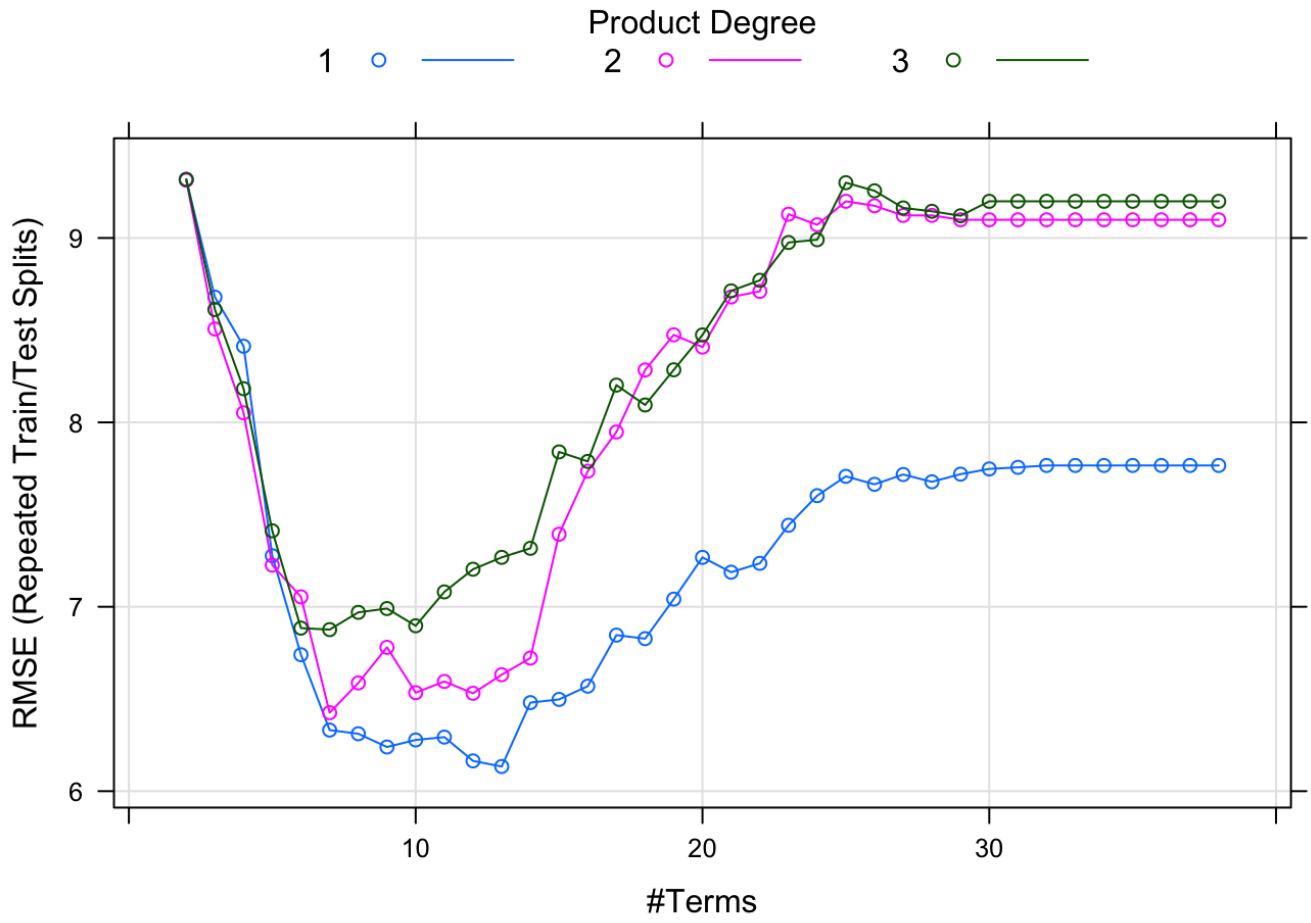
Mars Model with spatial sign removing correlated predictors

When I tried removing the correlated predictors for MARS model even with threshold of .999 I was getting only one predictors ie) first predictors. So I used that predictor to create the model

```
## Multivariate Adaptive Regression Spline
##
## 174 samples
## 1 predictor
##
## Pre-processing: spatial sign transformation (1), centered (1),
##  scaled (1)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 131, 131, 131, 131, 131, 131, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      9.317741  0.5351689  7.870507
##   1        3      8.678988  0.6019678  7.114238
##   1        4      8.412889  0.6271222  6.726207
##   1        5      7.276657  0.7140944  5.749177
##   1        6      6.740050  0.7567193  5.334827
```

```
##   1         7      6.331393  0.7787190  5.010476
##   1         8      6.311462  0.7836586  4.873798
##   1         9      6.239543  0.7910556  4.736754
##   1        10      6.277795  0.7924777  4.662157
##   1        11      6.293263  0.7956897  4.552791
##   1        12      6.164216  0.8033198  4.490451
##   1        13      6.133765  0.8041805  4.480577
##   1        14      6.480258  0.7894343  4.608146
##   1        15      6.497620  0.7901240  4.586948
##   1        16      6.569445  0.7877915  4.571849
##   1        17      6.846307  0.7784949  4.713419
##   1        18      6.827080  0.7801777  4.700819
##   1        19      7.041608  0.7763465  4.726124
##   1        20      7.267966  0.7682331  4.746072
##   1        21      7.187482  0.7754415  4.700468
##   1        22      7.235548  0.7737557  4.666711
##   1        23      7.442072  0.7659879  4.761031
##   1        24      7.602595  0.7568366  4.825240
##   1        25      7.707432  0.7503419  4.900240
##   1        26      7.663961  0.7534296  4.854452
##   1        27      7.717291  0.7499913  4.862226
##   1        28      7.677759  0.7527366  4.852260
##   1        29      7.719258  0.7520811  4.879503
##   1        30      7.747557  0.7511890  4.884614
##   1        31      7.756194  0.7513350  4.879819
##   1        32      7.766320  0.7509452  4.879599
##   1        33      7.766320  0.7509452  4.879599
##   1        34      7.766320  0.7509452  4.879599
##   1        35      7.766320  0.7509452  4.879599
##   1        36      7.766320  0.7509452  4.879599
##   1        37      7.766320  0.7509452  4.879599
##   1        38      7.766320  0.7509452  4.879599
##   2         2      9.313350  0.5355091  7.865379
##   2         3      8.506459  0.6087844  6.997120
##   2         4      8.051646  0.6568089  6.452389
##   2         5      7.225886  0.7135602  5.736960
##   2         6      7.053715  0.7387637  5.461354
##   2         7      6.425565  0.7770710  5.052338
##   2         8      6.587119  0.7697745  5.010831
##   2         9      6.780135  0.7517751  5.014991
##   2        10      6.534060  0.7698413  4.762005
##   2        11      6.595116  0.7665170  4.745479
##   2        12      6.530806  0.7700157  4.635058
##   2        13      6.631685  0.7672962  4.665281
##   2        14      6.721882  0.7654763  4.668350
##   2        15      7.393390  0.7365901  4.862095
##   2        16      7.735371  0.7226338  4.997909
##   2        17      7.947895  0.7163873  5.048605
##   2        18      8.284216  0.7002950  5.143749
##   2        19      8.474850  0.6918478  5.207155
##   2        20      8.408359  0.6969168  5.165804
##   2        21      8.680000  0.6888395  5.312695
##   2        22      8.710710  0.6879586  5.307105
##   2        23      9.129050  0.6750625  5.441920
##   2        24      9.071472  0.6809747  5.454506
##   2        25      9.199007  0.6751172  5.485130
##   2        26      9.174639  0.6759475  5.446234
```

```
##   2        26      9.174639  0.6759475  5.446234
##   2        27      9.122217  0.6763791  5.477959
##   2        28      9.122702  0.6763817  5.482021
##   2        29      9.098570  0.6769352  5.492579
##   2        30      9.098570  0.6769352  5.492579
##   2        31      9.098570  0.6769352  5.492579
##   2        32      9.098570  0.6769352  5.492579
##   2        33      9.098570  0.6769352  5.492579
##   2        34      9.098570  0.6769352  5.492579
##   2        35      9.098570  0.6769352  5.492579
##   2        36      9.098570  0.6769352  5.492579
##   2        37      9.098570  0.6769352  5.492579
##   2        38      9.098570  0.6769352  5.492579
##   3         2      9.317741  0.5351689  7.870507
##   3         3      8.611919  0.6092592  7.071216
##   3         4      8.182543  0.6512228  6.513665
##   3         5      7.411959  0.7031655  5.808509
##   3         6      6.884369  0.7440710  5.381163
##   3         7      6.876214  0.7480561  5.228560
##   3         8      6.969814  0.7378646  5.178367
##   3         9      6.990375  0.7319731  5.108348
##   3        10      6.896936  0.7399584  4.959672
##   3        11      7.080683  0.7301405  4.980706
##   3        12      7.203570  0.7246385  4.942440
##   3        13      7.268027  0.7258311  4.958551
##   3        14      7.317044  0.7316780  4.958194
##   3        15      7.839891  0.7113718  5.063260
##   3        16      7.788468  0.7144833  5.023893
##   3        17      8.202112  0.7026735  5.150106
##   3        18      8.094358  0.7078859  5.104357
##   3        19      8.285142  0.6995722  5.175227
##   3        20      8.474241  0.6931885  5.229380
##   3        21      8.713320  0.6828917  5.328935
##   3        22      8.770525  0.6800026  5.352506
##   3        23      8.975543  0.6783657  5.429922
##   3        24      8.990531  0.6805971  5.437165
##   3        25      9.299900  0.6754395  5.583085
##   3        26      9.255801  0.6793307  5.533188
##   3        27      9.163046  0.6810692  5.530968
##   3        28      9.144718  0.6804685  5.506871
##   3        29      9.120586  0.6810221  5.517428
##   3        30      9.198866  0.6819798  5.550163
##   3        31      9.198866  0.6819798  5.550163
##   3        32      9.198866  0.6819798  5.550163
##   3        33      9.198866  0.6819798  5.550163
##   3        34      9.198866  0.6819798  5.550163
##   3        35      9.198866  0.6819798  5.550163
##   3        36      9.198866  0.6819798  5.550163
##   3        37      9.198866  0.6819798  5.550163
##   3        38      9.198866  0.6819798  5.550163
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were nprune = 13 and degree = 1.
```



Test set

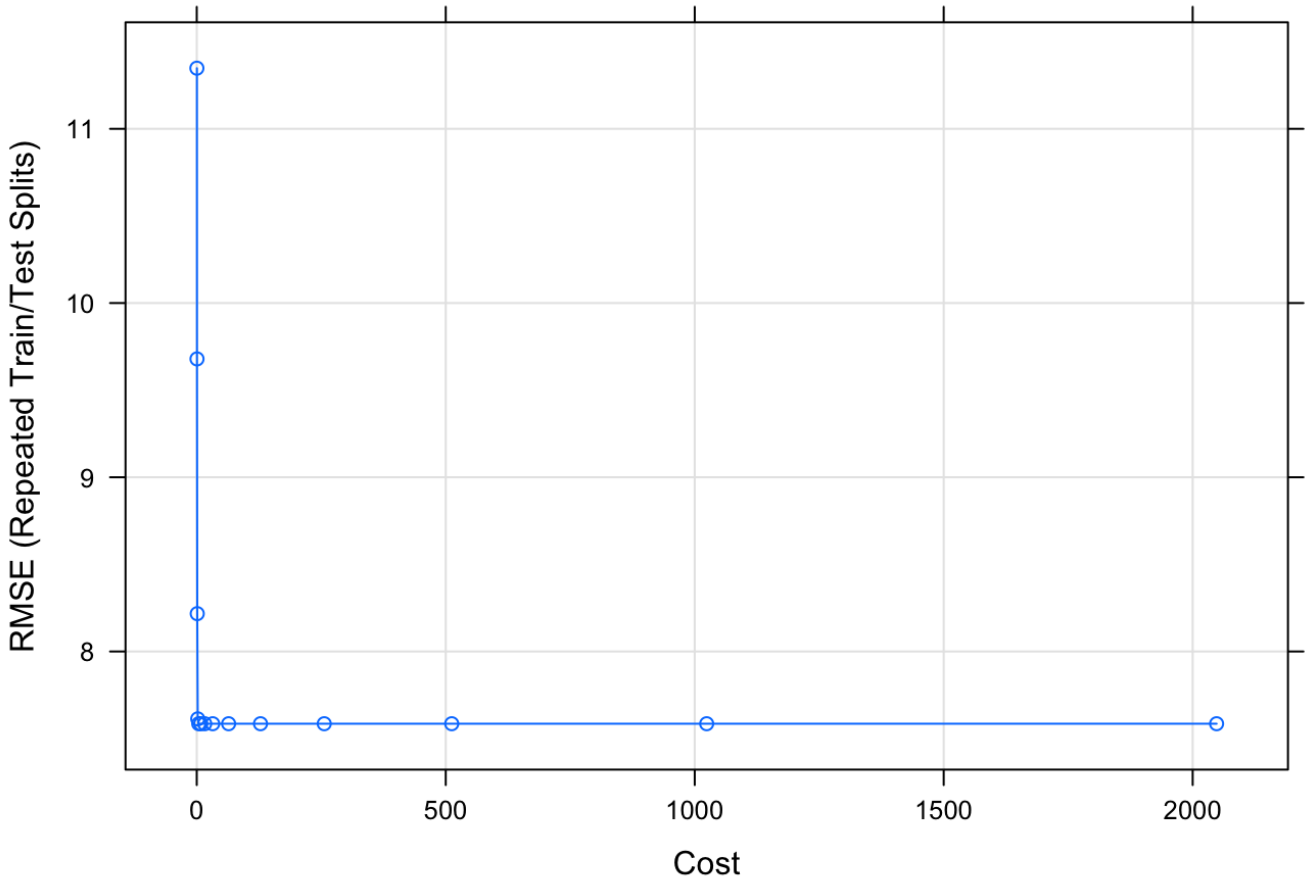
```
##      RMSE  Rsquared       MAE
## 5.3556324 0.8156073 3.9938803
```

From the above plot and the tabulated result, we can clearly see that the best model of mars model has nprune = 13 and degree = 1. The R^2 value obtained on the test set is 0.8156.

Support Vector Machine

Lets bulid the model using support vector machine using train() in caret package with svmRadial method which uses radial basis function. For svm, I set the tune length to be 14 as it tuneLength argument will use the default grid search of 20 cost values between 2^-2, 2^-1, . . . , 2^11. sigma is estimated analytically by default

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 131, 131, 131, 131, 131, 131, ...
## Resampling results across tuning parameters:
##
##    C        RMSE       Rsquared   MAE
##       0.25  11.348009  0.4953144  8.682498
##       0.50   9.679274  0.5886628  7.503213
##       1.00   8.217206  0.6529025  6.299173
##       2.00   7.613297  0.6873605  5.769243
##       4.00   7.585359  0.6884891  5.718025
##       8.00   7.585357  0.6884891  5.718000
##      16.00   7.585357  0.6884891  5.718000
##      32.00   7.585357  0.6884891  5.718000
##      64.00   7.585357  0.6884891  5.718000
##     128.00   7.585357  0.6884891  5.718000
##     256.00   7.585357  0.6884891  5.718000
##     512.00   7.585357  0.6884891  5.718000
##    1024.00   7.585357  0.6884891  5.718000
##    2048.00   7.585357  0.6884891  5.718000
##
## Tuning parameter 'sigma' was held constant at a value of 0.06775675
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were sigma = 0.06775675 and C = 8.
```



Test set

```
##      RMSE Rsquared      MAE
## 6.004810 0.705378 4.375796
```

From the above plot and the tabulated result, we can clearly see that the final values used for the model were sigma = 0.06775675 and C = 8. The R^2 value obtained on the test set is 0.7054.

Thus from buliding up all the above models, I have the following table.

Model	Parameter	Training RMSE	Training R Squared	Testing RMSE	Testing R Squared
KNN	k as 5	9.1336	0.5222	8.3322	0.45
Neural Network without PCA	size as 4 & decay as 0.01	0.8409	0.9954	9.5493	0.9987
Neural Network with PCA	size as 2 & decay as 0.01	12.0004	0.2054	9.5493	0.2661
Averaged Neural Network	size as 3 & decay as 0.01	0.6935	0.9973	0.3869	0.9989
Mars Model with no preprocessing	nprune as 24 & degree as	2.7309	0.9601	1.3921	0.985

Model	Parameter	Training RMSE	Training R Squared	Testing RMSE	Testing R Squared
Mars Model with spatial sign removing correlated predictors	2 nprune = 13 & degree = 1	6.1338	0.8042	5.3556	0.8156
Support Vector Machine	C as 8 & sigma as 0.06775675	7.5854	0.6885	6.0048	0.7054

So from the table we can see that the average neural network model gives best prediction as it has highest R^2 (0.9989) value on the test set. Coming back to the question of neural network sensitive to highly correlated predictors, we can clearly see that neurtal network without PCA (Test R square:- 0.9987) is performing better thsn the one with PCA (Test R square:- 0.2661).

Thus using proprocessing using PCA on neural does not helps.

Question 4

We will first load the dataset here in order to answer the further question. To ensure I have the data I checked the data dimension and printed out a small subset of the data.

```
##   X1 X2 X3 X4 X5
## 1  0  0  0  0  0
## 2  0  0  0  0  0
## 3  0  0  0  0  0
## 4  0  0  0  0  0
## 5  0  0  0  0  0
## 6  0  0  0  0  0
```

```
## [1] 12.520  1.120 19.405  1.730  1.680
```

Thus we see that both the predictors and target values are loaded properly. The fingerprints contains the 1107 binary molecular predictors for the 165 compounds, while permeability contains permeability response is the responding variable. It is been told in Exercise 6.2 that the fingerprint predictors are having sparse data. In order to handle this we are going to apply near zero variance to handle this using caret package.

```
## [1] "Number of predictors left out for modeling is 388"
```

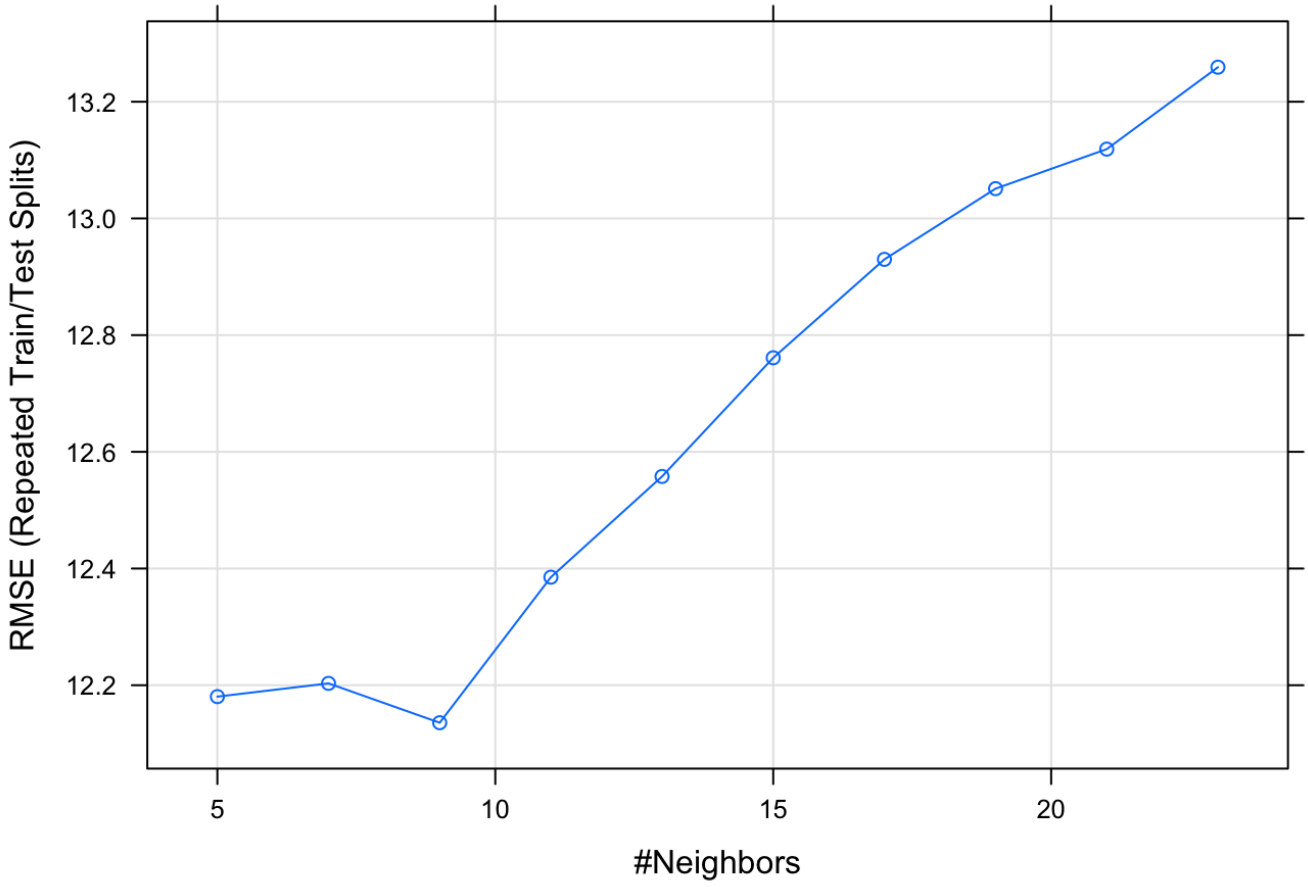
Thus after removing the near zero variance from the model we have 388 predictors left our in the data. We will use this predictors to bulid the models.

To split the data into a training and a test set, I have 80% of the data put into the train set and left out data samples in the train set. Then using the splitted train set, I will use Leave group out cross validation because the size of the data points is considerably greater than the size of the predictors. Lets bulid the models.

KNN

Lets first fit the data with a KNN model. For this I have used train() in the caret package with knn as method. Below we have complete information about the KNN model.

```
## k-Nearest Neighbors
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  12.18047  0.4193401   8.328473
##    7  12.20318  0.4130314   8.443593
##    9  12.13573  0.4234838   8.385506
##   11  12.38515  0.3965554   8.666109
##   13  12.55781  0.3782623   8.902542
##   15  12.76116  0.3539242   9.259150
##   17  12.92986  0.3381347   9.560620
##   19  13.05106  0.3223688   9.796124
##   21  13.11881  0.3178781   9.944690
##   23  13.25921  0.3072111  10.108553
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was k = 9.
```



Test set

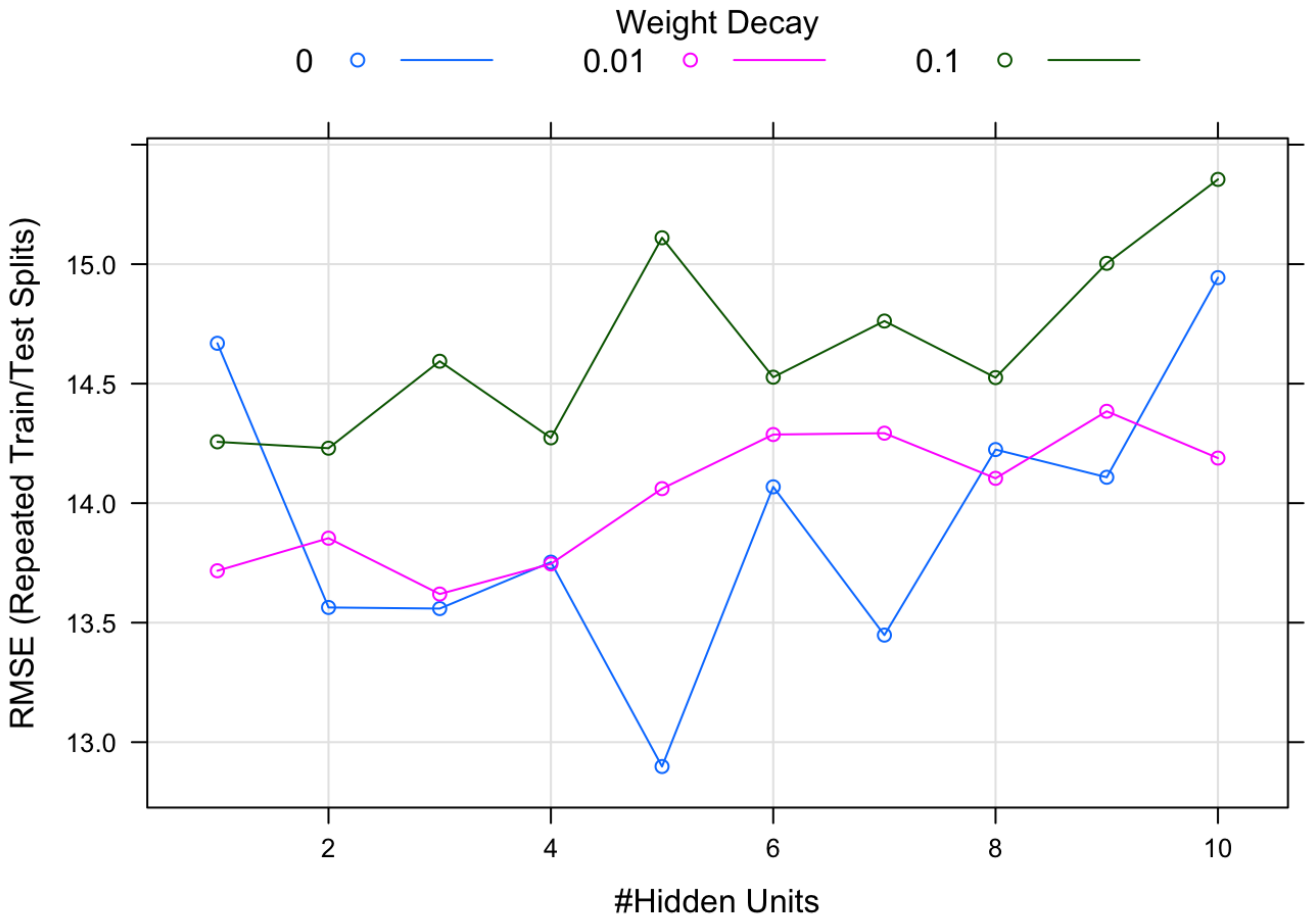
```
##       RMSE   Rsquared        MAE
## 11.5584796  0.3785895  8.1628793
```

From the above plot and the tabulated result, we can clearly see that the best model has the k value of 9. That is the best R^2 is obtain when we are considering 9 nearest neighbours. The R^2 value obtained on the test set is 0.3786.

Neural Network

Lets bulid the model using neural network model using train() in caret package with method as nnet. I am setting the decay to be 0, 0.01, 0.1 and size varying from 1 to 10.

```
## Neural Network
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE      Rsquared   MAE
##   0.00    1    14.66894  0.1992654  11.007925
##   0.00    2    13.56363  0.3050131   9.964243
##   0.00    3    13.55893  0.3519219   9.954022
##   0.00    4    13.75327  0.3448457  10.082438
##   0.00    5    12.89814  0.4091014   9.677077
##   0.00    6    14.06830  0.3396351  10.564821
##   0.00    7    13.44795  0.3951566   9.961410
##   0.00    8    14.22428  0.3524452  10.497537
##   0.00    9    14.10839  0.3690021  10.373561
##   0.00   10    14.94362  0.3480160  10.998434
##   0.01    1    13.71719  0.3845546   9.113970
##   0.01    2    13.85347  0.3638802  10.007150
##   0.01    3    13.61979  0.3678061  10.103614
##   0.01    4    13.74590  0.3540830  10.212034
##   0.01    5    14.06074  0.3668699  10.442635
##   0.01    6    14.28716  0.3564290  10.629968
##   0.01    7    14.29278  0.3408776  10.756775
##   0.01    8    14.10389  0.3753960  10.443204
##   0.01    9    14.38443  0.3634419  10.617678
##   0.01   10    14.18870  0.3862505  10.663720
##   0.10    1    14.25657  0.3672503   9.443107
##   0.10    2    14.22994  0.3649989  10.397407
##   0.10    3    14.59384  0.3576527  10.731541
##   0.10    4    14.27325  0.3499417  10.435441
##   0.10    5    15.11034  0.3132665  11.138925
##   0.10    6    14.52704  0.3517566  10.853295
##   0.10    7    14.76204  0.3475786  10.971939
##   0.10    8    14.52536  0.3483075  10.905612
##   0.10    9    15.00350  0.3220190  11.066886
##   0.10   10    15.35468  0.3276643  11.333168
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were size = 5 and decay = 0.
```



Test set

```
##       RMSE   Rsquared        MAE
## 14.2702625  0.2914559 11.1760107
```

From the above plot and the tabulated result, we can clearly see that the best model of neural network model is of size 5 and decay is 0. The R^2 value obtained on the test set is 0.2915.

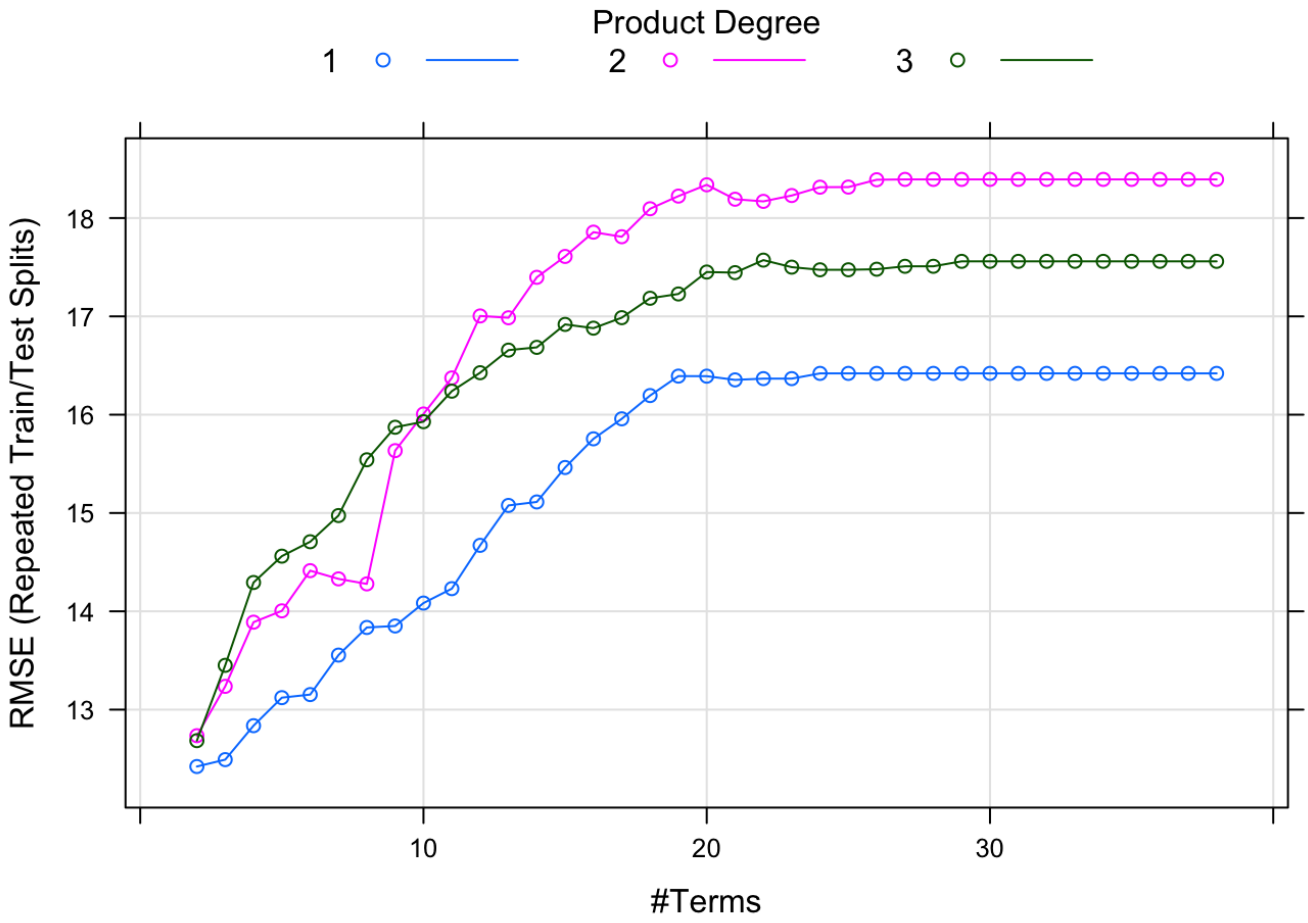
Mars Model with no preprocessing

Lets bulid the model using mars model using train() in caret package with method as earth. I am setting the degree to be 1, 2, 3 and number of prune varying from 2 to 38. I tried applying spatial sign for preprocess which gave me worst prediction on the test set with 0.23 R square value. Thus I used center and scaling as preprocess step for the Mars model and below the details.

```
## Multivariate Adaptive Regression Spline
##
## 133 samples
## 388 predictors
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      12.42146  0.4099226   9.071569
##   1        3      12.49142  0.4018141   8.570705
##   1        4      12.83669  0.3769528   8.739432
##   1        5      13.12187  0.3594738   8.986555
```

```
##   1         5      13.12187  0.3594758   8.986555
##   1         6      13.15300  0.3661011   9.248789
##   1         7      13.55472  0.3617997   9.734635
##   1         8      13.83565  0.3647982   9.751219
##   1         9      13.85085  0.3657400   9.617669
##   1        10      14.08389  0.3579442   9.765826
##   1        11      14.23037  0.3506204   9.934069
##   1        12      14.67098  0.3320251  10.153010
##   1        13      15.07703  0.3243381  10.438316
##   1        14      15.11239  0.3224550  10.513071
##   1        15      15.46279  0.3116091  10.706421
##   1        16      15.75441  0.3118065  10.880115
##   1        17      15.95799  0.3136765  11.005202
##   1        18      16.19448  0.3090689  11.121403
##   1        19      16.39237  0.3083221  11.231873
##   1        20      16.39173  0.3095404  11.248110
##   1        21      16.35396  0.3119618  11.204453
##   1        22      16.36672  0.3114318  11.218035
##   1        23      16.36672  0.3114318  11.218035
##   1        24      16.42006  0.3102059  11.255753
##   1        25      16.42006  0.3102059  11.255753
##   1        26      16.42006  0.3102059  11.255753
##   1        27      16.42006  0.3102059  11.255753
##   1        28      16.42006  0.3102059  11.255753
##   1        29      16.42006  0.3102059  11.255753
##   1        30      16.42006  0.3102059  11.255753
##   1        31      16.42006  0.3102059  11.255753
##   1        32      16.42006  0.3102059  11.255753
##   1        33      16.42006  0.3102059  11.255753
##   1        34      16.42006  0.3102059  11.255753
##   1        35      16.42006  0.3102059  11.255753
##   1        36      16.42006  0.3102059  11.255753
##   1        37      16.42006  0.3102059  11.255753
##   1        38      16.42006  0.3102059  11.255753
##   2         2      12.73476  0.3810779   9.279446
##   2         3      13.23672  0.3374476   9.174074
##   2         4      13.88950  0.3161362   9.655913
##   2         5      14.00487  0.3202137   9.624312
##   2         6      14.41363  0.3175609   9.743502
##   2         7      14.32989  0.3219546   9.576293
##   2         8      14.27880  0.3291617   9.578573
##   2         9      15.63422  0.2778098  10.123078
##   2        10      16.00711  0.2706481  10.376437
##   2        11      16.37365  0.2610798  10.529966
##   2        12      17.00425  0.2471498  10.807932
##   2        13      16.98567  0.2475079  10.885248
##   2        14      17.39728  0.2412196  11.223152
##   2        15      17.60951  0.2391672  11.336776
##   2        16      17.85653  0.2353091  11.391346
##   2        17      17.80994  0.2411536  11.460800
##   2        18      18.09462  0.2282308  11.646935
##   2        19      18.22320  0.2216317  11.705319
##   2        20      18.33814  0.2176238  11.690920
##   2        21      18.19138  0.2250705  11.605744
##   2        22      18.16959  0.2255261  11.610091
##   2        23      18.22906  0.2250847  11.621930
##   2        24      18.31406  0.2249958  11.653615
```

```
##   2       25      18.31501  0.2248615  11.663328
##   2       26      18.38967  0.2215299  11.693625
##   2       27      18.39323  0.2215028  11.712044
##   2       28      18.39323  0.2215028  11.712044
##   2       29      18.39323  0.2215028  11.712044
##   2       30      18.39323  0.2215028  11.712044
##   2       31      18.39323  0.2215028  11.712044
##   2       32      18.39323  0.2215028  11.712044
##   2       33      18.39323  0.2215028  11.712044
##   2       34      18.39323  0.2215028  11.712044
##   2       35      18.39323  0.2215028  11.712044
##   2       36      18.39323  0.2215028  11.712044
##   2       37      18.39323  0.2215028  11.712044
##   2       38      18.39323  0.2215028  11.712044
##   3        2      12.68434  0.3873686   9.237108
##   3        3      13.45065  0.3229941   9.300490
##   3        4      14.29311  0.2862546   9.622268
##   3        5      14.56109  0.2848627   9.724570
##   3        6      14.70754  0.2805936   9.547797
##   3        7      14.97412  0.2909159   9.649438
##   3        8      15.54115  0.2690676   9.922469
##   3        9      15.87209  0.2611280  10.169324
##   3       10      15.92877  0.2628880  10.134016
##   3       11      16.23914  0.2496414  10.273357
##   3       12      16.42686  0.2496481  10.307985
##   3       13      16.65665  0.2472425  10.386725
##   3       14      16.68472  0.2525662  10.353012
##   3       15      16.91827  0.2468946  10.423964
##   3       16      16.87999  0.2553404  10.390833
##   3       17      16.98664  0.2587741  10.431291
##   3       18      17.18408  0.2539071  10.582685
##   3       19      17.22770  0.2580855  10.605322
##   3       20      17.45099  0.2503162  10.701284
##   3       21      17.44492  0.2461660  10.632780
##   3       22      17.57205  0.2450834  10.676506
##   3       23      17.50062  0.2486290  10.678435
##   3       24      17.47367  0.2497434  10.636471
##   3       25      17.47385  0.2495463  10.653408
##   3       26      17.47962  0.2495825  10.652147
##   3       27      17.50926  0.2480192  10.678255
##   3       28      17.50926  0.2480192  10.678255
##   3       29      17.55967  0.2454125  10.755039
##   3       30      17.55967  0.2454125  10.755039
##   3       31      17.55967  0.2454125  10.755039
##   3       32      17.55967  0.2454125  10.755039
##   3       33      17.55967  0.2454125  10.755039
##   3       34      17.55967  0.2454125  10.755039
##   3       35      17.55967  0.2454125  10.755039
##   3       36      17.55967  0.2454125  10.755039
##   3       37      17.55967  0.2454125  10.755039
##   3       38      17.55967  0.2454125  10.755039
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were nprune = 2 and degree = 1.
```



Test set

```
##      RMSE  Rsquared       MAE
## 9.8692942 0.5469159 7.4774994
```

From the above plot and the tabulated result, we can clearly see that the best model of mars model has nprune = 2 and degree = 1. The R^2 value obtained on the test set is 0.5469.

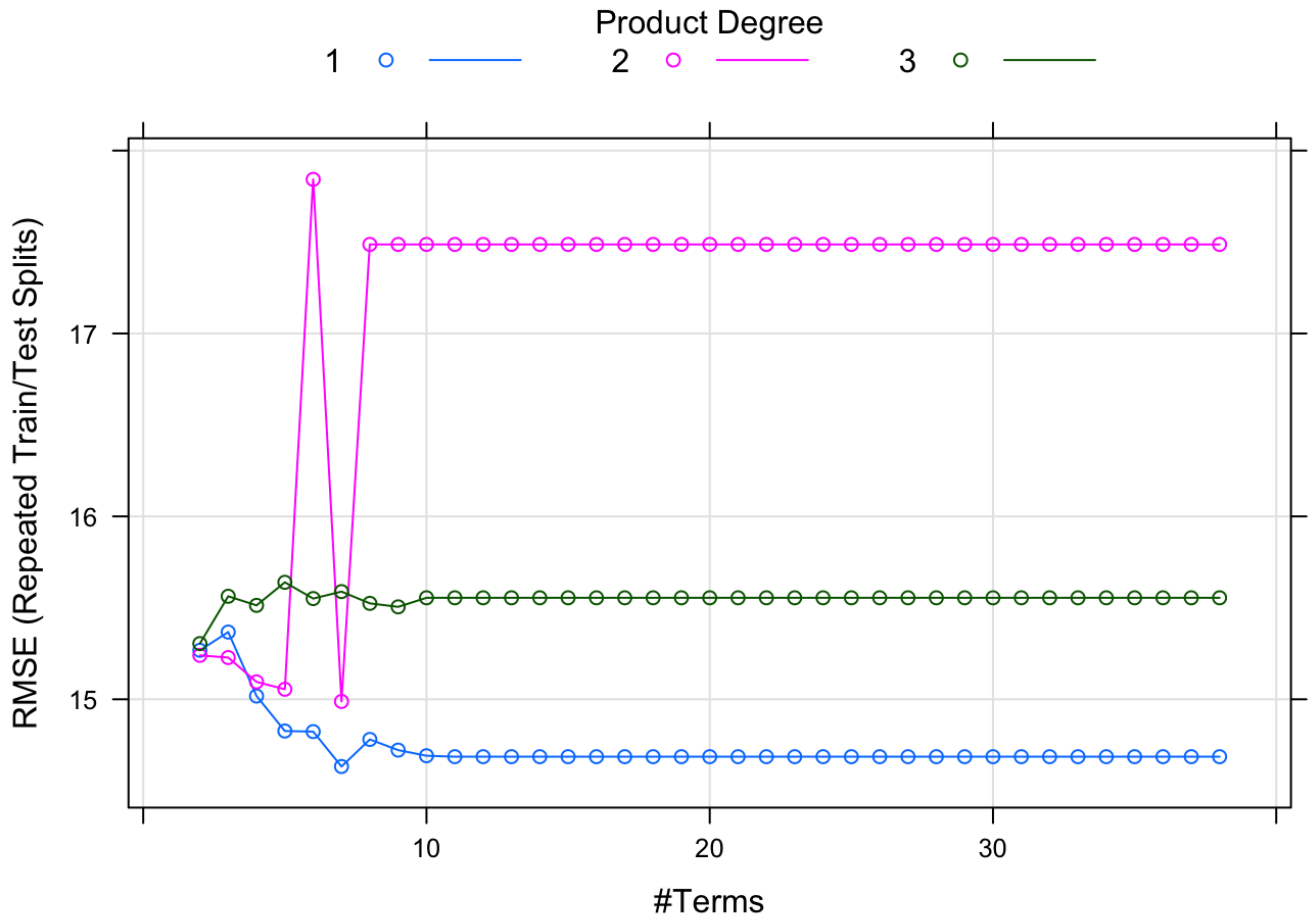
Mars Model removing correlation

Lets bulid the model using mars model using train() in caret package with method as earth. With a small change here that we will remove the correlated predictors out and then train the model. I am setting the degree to be 1, 2, 3 and number of prune varying from 2 to 38.

```
## Multivariate Adaptive Regression Spline
##
## 133 samples
##  83 predictor
##
## Pre-processing: centered (83), scaled (83)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      15.26772  0.1209577  11.79985
##   1        3      15.36755  0.1138346  11.56263
##   1        4      15.01790  0.1388334  11.24805
##   1        5      14.82656  0.1549963  11.02253
##   1        6      14.82341  0.1647039  11.09393
```

```
##   1        7      14.63252  0.1883691  10.98275
##   1        8      14.78057  0.1829358  11.07865
##   1        9      14.72215  0.1893190  11.08294
##   1       10      14.69136  0.1894610  11.07120
##   1       11      14.68626  0.1903254  11.02948
##   1       12      14.68626  0.1903254  11.02948
##   1       13      14.68626  0.1903254  11.02948
##   1       14      14.68626  0.1903254  11.02948
##   1       15      14.68626  0.1903254  11.02948
##   1       16      14.68626  0.1903254  11.02948
##   1       17      14.68626  0.1903254  11.02948
##   1       18      14.68626  0.1903254  11.02948
##   1       19      14.68626  0.1903254  11.02948
##   1       20      14.68626  0.1903254  11.02948
##   1       21      14.68626  0.1903254  11.02948
##   1       22      14.68626  0.1903254  11.02948
##   1       23      14.68626  0.1903254  11.02948
##   1       24      14.68626  0.1903254  11.02948
##   1       25      14.68626  0.1903254  11.02948
##   1       26      14.68626  0.1903254  11.02948
##   1       27      14.68626  0.1903254  11.02948
##   1       28      14.68626  0.1903254  11.02948
##   1       29      14.68626  0.1903254  11.02948
##   1       30      14.68626  0.1903254  11.02948
##   1       31      14.68626  0.1903254  11.02948
##   1       32      14.68626  0.1903254  11.02948
##   1       33      14.68626  0.1903254  11.02948
##   1       34      14.68626  0.1903254  11.02948
##   1       35      14.68626  0.1903254  11.02948
##   1       36      14.68626  0.1903254  11.02948
##   1       37      14.68626  0.1903254  11.02948
##   1       38      14.68626  0.1903254  11.02948
##   2        2      15.24019  0.1203556  11.76977
##   2        3      15.22822  0.1251379  11.52902
##   2        4      15.09519  0.1275473  11.30143
##   2        5      15.05492  0.1310002  11.22278
##   2        6      17.84249  0.1311944  11.73136
##   2        7      14.98824  0.1442769  11.14337
##   2        8      17.48682  0.1401701  11.64996
##   2        9      17.48682  0.1401701  11.64996
##   2       10      17.48682  0.1401701  11.64996
##   2       11      17.48682  0.1401701  11.64996
##   2       12      17.48682  0.1401701  11.64996
##   2       13      17.48682  0.1401701  11.64996
##   2       14      17.48682  0.1401701  11.64996
##   2       15      17.48682  0.1401701  11.64996
##   2       16      17.48682  0.1401701  11.64996
##   2       17      17.48682  0.1401701  11.64996
##   2       18      17.48682  0.1401701  11.64996
##   2       19      17.48682  0.1401701  11.64996
##   2       20      17.48682  0.1401701  11.64996
##   2       21      17.48682  0.1401701  11.64996
##   2       22      17.48682  0.1401701  11.64996
##   2       23      17.48682  0.1401701  11.64996
##   2       24      17.48682  0.1401701  11.64996
##   2       25      17.48682  0.1401701  11.64996
```

```
##   2       26      17.48682  0.1401701  11.64996
##   2       27      17.48682  0.1401701  11.64996
##   2       28      17.48682  0.1401701  11.64996
##   2       29      17.48682  0.1401701  11.64996
##   2       30      17.48682  0.1401701  11.64996
##   2       31      17.48682  0.1401701  11.64996
##   2       32      17.48682  0.1401701  11.64996
##   2       33      17.48682  0.1401701  11.64996
##   2       34      17.48682  0.1401701  11.64996
##   2       35      17.48682  0.1401701  11.64996
##   2       36      17.48682  0.1401701  11.64996
##   2       37      17.48682  0.1401701  11.64996
##   2       38      17.48682  0.1401701  11.64996
##   3        2      15.30468  0.1225605  11.80172
##   3        3      15.56329  0.1130472  11.79292
##   3        4      15.51354  0.1006252  11.63641
##   3        5      15.63962  0.1115818  11.69784
##   3        6      15.55068  0.1211300  11.59151
##   3        7      15.58878  0.1232556  11.59138
##   3        8      15.52450  0.1270505  11.57711
##   3        9      15.50542  0.1288293  11.56971
##   3       10      15.55495  0.1266223  11.58064
##   3       11      15.55495  0.1266223  11.58064
##   3       12      15.55495  0.1266223  11.58064
##   3       13      15.55495  0.1266223  11.58064
##   3       14      15.55495  0.1266223  11.58064
##   3       15      15.55495  0.1266223  11.58064
##   3       16      15.55495  0.1266223  11.58064
##   3       17      15.55495  0.1266223  11.58064
##   3       18      15.55495  0.1266223  11.58064
##   3       19      15.55495  0.1266223  11.58064
##   3       20      15.55495  0.1266223  11.58064
##   3       21      15.55495  0.1266223  11.58064
##   3       22      15.55495  0.1266223  11.58064
##   3       23      15.55495  0.1266223  11.58064
##   3       24      15.55495  0.1266223  11.58064
##   3       25      15.55495  0.1266223  11.58064
##   3       26      15.55495  0.1266223  11.58064
##   3       27      15.55495  0.1266223  11.58064
##   3       28      15.55495  0.1266223  11.58064
##   3       29      15.55495  0.1266223  11.58064
##   3       30      15.55495  0.1266223  11.58064
##   3       31      15.55495  0.1266223  11.58064
##   3       32      15.55495  0.1266223  11.58064
##   3       33      15.55495  0.1266223  11.58064
##   3       34      15.55495  0.1266223  11.58064
##   3       35      15.55495  0.1266223  11.58064
##   3       36      15.55495  0.1266223  11.58064
##   3       37      15.55495  0.1266223  11.58064
##   3       38      15.55495  0.1266223  11.58064
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were nprune = 7 and degree = 1.
```



Test set

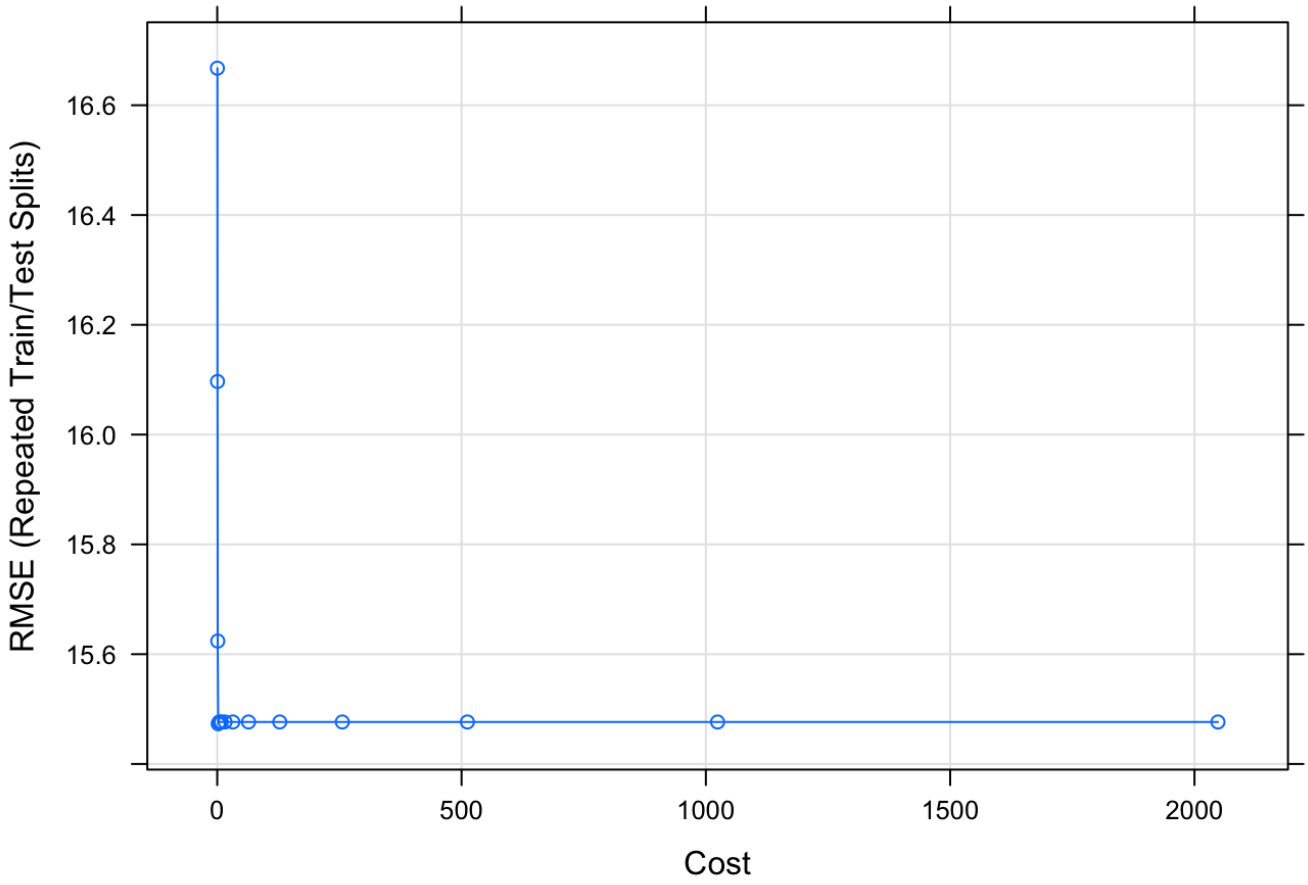
```
##        RMSE    Rsquared         MAE
## 14.1887564  0.1371695 11.0669611
```

From the above plot and the tabulated result, we can clearly see that the best model of mars model has nprune = 7 and degree = 1. The R^2 value obtained on the test set is 0.1372.

Support Vector Machine

Lets bulid the model using support v ector machine using train() in caret package with svmRadial method which uses radial basis function. For svm, I set the tune length to be 14 as it tuneLength argument will use the default grid search of 20 cost values between 2^-2, 2^-1, . . . , 2^11. sigma is estimated analytically by default

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   C        RMSE      Rsquared    MAE
##      0.25  16.66748  0.01417462  10.59983
##      0.50  16.09675  0.03861270  10.87586
##      1.00  15.62397  0.04326156  11.39807
##      2.00  15.47332  0.02000953  12.20375
##      4.00  15.47650  0.02009189  12.29866
##      8.00  15.47650  0.02009189  12.29866
##     16.00  15.47650  0.02009189  12.29866
##     32.00  15.47650  0.02009189  12.29866
##     64.00  15.47650  0.02009189  12.29866
##    128.00  15.47650  0.02009189  12.29866
##    256.00  15.47650  0.02009189  12.29866
##    512.00  15.47650  0.02009189  12.29866
##   1024.00  15.47650  0.02009189  12.29866
##   2048.00  15.47650  0.02009189  12.29866
##
## Tuning parameter 'sigma' was held constant at a value of 0.002329905
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were sigma = 0.002329905 and C = 2.
```



Test set

```
##        RMSE    Rsquared         MAE
## 11.7033030   0.3989128   9.4466225
```

From the above plot and the tabulated result, we can clearly see that the final values used for the model were sigma = 0.00233 and C = 2. The R^2 value obtained on the test set is 0.3989.

Thus from buliding up all the above models, I have the following table.

Model	Parameter	Training RMSE	Training R Squared	Testing RMSE	Testing R Squared
KNN	k as 9	12.1357	0.4234	11.5585	0.3786
Neural Network	size as 5 & decay as 0	12.8981	0.4091	14.2703	0.2915
MARS with no preprocessing	nprune as 2 & degree as 2	12.4215	0.4099	9.8693	0.5469
MARS Removing correlation	nprune as 7 & degree as 1	14.6325	0.1884	14.1888	0.1372
Support Vector Machine	C as 2 & sigma as 0.00233	15.4733	0.0200	11.7033	0.3989

Section (a)

From the above table we can clearly see that MARS model is performing both on optimal resampling and test set with comparatively high R square value as 0.4099 on training and 0.5469 on the testing set.

Section (b)

Yes, MARS model performs better than the optimal linear model which is a Partial Least Square model with 7 components (Test set R square: 0.4457). MARS model has the test set R square value of 0.5469. This shows that the underlying relationship between the predictors and responding variable is non linear.

Section (c)

No, I won't recommend the model because the best model MARS itself has a test set R square value of 0.5469 which quite on the lower end.

*** End of Solution ***

Appendix - Coding

installing the packages

installNewPackage <- function(packageName) {

```
    if(packageName  %in% rownames(installed.packages()) == FALSE)

    {

            install.packages(packageName, repos = "http://cran.us.r-project.org", d
ependencies=TRUE)

    }
```

}

installNewPackage("kernlab")

installNewPackage("mlbench")

installNewPackage("caret")

installNewPackage("AppliedPredictiveModeling")

library(kernlab)

library(mlbench)

library(caret)

library(AppliedPredictiveModeling)

Question 1

Code from the book

set.seed(1)

x <- runif(100, min = 2, max = 10)

```
y <- sin(x) + rnorm(length(x)) * .25

sinData <- data.frame(x = x, y = y)

plot(x, y)

## Create a grid of x values to use for prediction

dataGrid <- data.frame(x = seq(2, 10, length = 100))

head(dataGrid)
```

Section (a)

```
# Setting the expand grid for the cost and epsilon parameters just like the last Home work

svm_parameter_grid <- expand.grid(costs = 2^c(-2, 0, 2, 8), epsilons = c(.01, .05, .1, .5))

# Show the grid used

svm_parameter_grid

# Intializing the empty result dataframe

result_df <- data.frame(actual = double(), pred = double(), costs = factor(), epsilons = factor())

# Set the screen split

par(mfrow = c(4, 4))

# Looping over the combination of SVM parameter grid

for(index in 1:nrow(svm_parameter_grid)) {
```

```
\# Setting the seed

set.seed(1)

\# Buliding the SVM

rbf_model <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", kpar = "automatic
", C = svm_parameter_grid$costs[index],

               epsilon = svm_parameter_grid$epsilons[index])


\# Make the prediction using the model generated

prediction <- predict(rbf_model, newdata = dataGrid)

plot(sinData$x, sinData$y, main = paste("Cost:", toString(svm_parameter_grid$costs[
index]), "| Epsilon:", toString(svm_parameter_grid$epsilons[index])),

     xlab = "x", ylab = "y", cex = 2)

points(dataGrid$x, prediction, type = "l", col = "red", lwd = 3)

\# Create the this iteration result data frame

iter_results <- data.frame(actual = dataGrid$x, pred = prediction, costs = paste("C
ost:", toString(svm_parameter_grid$costs[index])),

                                                    epsilons = paste("Epsilo
n:", toString(svm_parameter_grid$epsilons[index])))

result_df <- rbind(result_df, iter_results)
```

}

Section (b)

Setting seed for expand grid

set.seed(1)

Setting the expand grid for the cost and epsilon parameters just like the last Home work

svm_parameter_grid <- expand.grid(costs = 2^c(-2, 0, 2, 8), epsilons = c(.01, .05, .1, .5))

sigma_values <- data.frame(sigma = c(0.2043268, 0.9977490, 47.74), color = c(“red”, “blue”, “green”))

Intializing the empty result dataframe

result_df <- data.frame(actual = double(), pred = double(), costs = factor(), epsilons = factor())

Set the screen split

par(mfrow = c(4, 4))

Looping over the combination of SVM parameter grid

for(index in 1:nrow(svm_parameter_grid)) {

```
plot(sinData$x, sinData$y, main = paste("Cost:", toString(svm_parameter_grid$costs[
index]), "| Epsilon:", toString(svm_parameter_grid$epsilons[index])),

     xlab = "x", ylab = "y", col = "black", lwd = 2)

for(i in 1:nrow(sigma_values)) {

    \# Setting the seed

    set.seed(1)

    \# Buliding the SVM

    rbf_model <- ksvm(x = x, y = y, data = sinData, kernel ="rbfdot", C = svm_param
eter_grid$costs[index],

                      epsilon = svm_parameter_grid$epsilons[index], kpar = list(sigma
= sigma_values$sigma[i]))

    \# Make the prediction using the model generated

    prediction <- predict(rbf_model, newdata = dataGrid)

    points(dataGrid$x, prediction, type = "l", col = sigma_values$color[i], lwd = 3
)

    \# Create the this iteration result data frame

    iter_results <- data.frame(actual = dataGrid$x, pred = prediction, costs = past
e("Cost:", toString(svm_parameter_grid$costs[index])),

                                                                epsilons = paste("Ep
silon:", toString(svm_parameter_grid$epsilons[index])))

    result_df <- rbind(result_df, iter_results)

}
```

}

par(mai=c(0,0,0,0))

plot.new()

legend(x = “center”, legend = sigma_values$sigma$, $col = sigma_values$color, lwd=4, cex=3, title = “Sigma Values”, horiz = TRUE)

Question 2

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(trainingDatax)

Look at the data using

featurePlot(trainingDatax, trainingDatay)

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(testDatax)

KNN

Set the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the KNN model

if(file.exists(“models/knn_model_q2.rds”)) {

```
knn_model <- readRDS("models/knn_model_q2.rds")
```

} else {

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

saveRDS(knn_model, "models/knn_model_q2.rds")
```

}

Print the model

knn_model

Plot the model

plot(knn_model)

Predict the model

knn_pred <- predict(knn_model, newdata = testData$x)

Get the test Set performance metrics

postResample(pred = knn_pred, obs = testData$y)

Neural Network

Create the grid for the network

nn_grid <- expand.grid(.decay = c(0, 0.01, 0.1), .size = 1:10)

Set the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/nnet_model_q2.rds”)) {

```
nnet_model <- readRDS("models/nnet_model_q2.rds")
```

} else {

```
nnet_model <- train(x = trainingData$x, y = trainingData$y, tuneGrid = nn_grid, met
hod = "nnet", preProc = c("center", "scale"),

               linout = TRUE, trace = FALSE, MaxNWts = 10 * (ncol(trainingData$x)+1
) + 10 + 1, maxit=500)

saveRDS(nnet_model, "models/nnet_model_q2.rds")
```

}

Print the model

nnet_model

Plot the model

plot(nnet_model)

Predict the test set

nnet_pred <- predict(nnet_model, newdata = testData$x)

Get the test Set performance metrics

postResample(pred = nnet_pred, obs = testData$y)

Averaged Neural Network

Create the tune grid

tune_grid <- expand.grid(.decay = c(0, 0.01, .1), .size = c(1:10), .bag = FALSE)

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/avg_nnet_model_q2.rds”)) {

```
avg_nnet_model <- readRDS("models/avg_nnet_model_q2.rds")
```

} else {

```
avg_nnet_model <- train(x = trainingData$x, y = trainingData$y, tuneGrid = tune_gri
d, method = "avNNet", preProc = c("center", "scale"),

                linout = TRUE, trace = FALSE, MaxNWts = 10 * (ncol(trainingData$x)
+ 1) + 10 + 1, maxit = 500)

saveRDS(avg_nnet_model, "models/avg_nnet_model_q2.rds")
```

}

Print the model

avg_nnet_model

Plot the model

plot(avg_nnet_model)

Make the prediction

avg_nnet_pred <- predict(avg_nnet_model, newdata = testData$x)

Get the performance scores

postResample(pred = avg_nnet_pred, obs = testData$y)

Mars Model with no Preprocessing

Create the tune grid

tune_grid <- expand.grid(.degree = 1:3, .nprune = 2:38)

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/mars_model_q2.rds”)) {

```
mars_model <- readRDS("models/mars_model_q2.rds")
```

} else {

```
mars_model <- train(x = trainingData$x, y = trainingData$y, tuneGrid = tune_grid, m
ethod = "earth")

saveRDS(mars_model, "models/mars_model_q2.rds")
```

}

Print the model

mars_model

Plot the model

plot(mars_model)

Make the prediction

mars_pred <- predict(mars_model, newdata = testData$x)

Get the performance scores

postResample(pred = mars_pred, obs = testData$y)

MARS with spatial sign and correlated predictors removed

Remove the highly correlated value

highlyCorDescr <- findCorrelation(cor(trainingData$x), cutoff = .75)

trainingCorData <- trainingData$x[,-highlyCorDescr]

Dimension of the predictors data

print(paste("No of highly correlated of predictors", dim(trainingCorData)[2]))

Create the tune grid

tune_grid <- expand.grid(.degree = 1:3, .nprune = 2:38)

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists("models/mars_model_Spatial_q2.rds")) {

```
mars_model1 <- readRDS("models/mars_model_Spatial_q2.rds")
```

} else {

```
mars_model1 <- train(x = trainingData$x, y = trainingData$y, preProcess = "spatialS
ign", tuneGrid = tune_grid, method = "earth")

saveRDS(mars_model, "models/mars_model_Spatial_q2.rds")
```

}

Print the model

mars_model1

Plot the model

plot(mars_model1)

Make the prediction

mars_pred <- predict(mars_model1, newdata = testData$x)

Get the performance scores

postResample(pred = mars_pred, obs = testData$y)

Support Vector Machine

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/svm_model_q2.rds”)) {

```
svm_model <- readRDS("models/svm_model_q2.rds")
```

} else {

```
svm_model <- svm_model <- train(x = trainingData$x, y = trainingData$y, tuneLength
= 14, method = "svmRadial", preProc = c("center","scale"))

saveRDS(svm_model, "models/svm_model_q2.rds")
```

}

Print the model

svm_model

Plot the model

plot(svm_model)

Make the prediction

svm_pred <- predict(svm_model, newdata = testData$x)

Get the performance scores

postResample(pred = svm_pred, obs = testData$y)

mars_mode_imp <- varImp(mars_model)

plot(mars_mode_imp, top = 10)

Question 3

Load the data

data(tecator)

absorp_df <- as.data.frame(absorp)

endpoints_df <- as.data.frame(endpoints)

colnames(endpoints_df) = c(“mositure”, “fat”,“protein”)

Verify the data is loaded or not

head(absorp_df[1:5])

head(endpoints_df)

Setting the seed for reproduciablity

set.seed(1)

Performing data spliting

cv_index <- createDataPartition(endpoints[, 2], p = 0.8, list = FALSE)

absorpTrain <- absorp_df[cv_index,]

absorpTest <- absorp_df[-cv_index,]

yTrain <- endpoints_df[cv_index, 2]

yTest <- endpoints_df[-cv_index, 2]

Setting up the control parameter

ctrl <- trainControl(method = “LGOCV”, repeats = 5)

KNN

Set the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the KNN model

if(file.exists(“models/knn_model_q3.rds”)) {

```
knn_model <- readRDS("models/knn_model_q3.rds")
```

} else {

```
knn_model <- train(x = absorpTrain, y = yTrain, method = "knn", preProc = c("center
", "scale"), tuneLength = 10)

saveRDS(knn_model, "models/knn_model_q3.rds")
```

}

Print the model

knn_model

Plot the model

plot(knn_model)

Predict the model

knn_pred <- predict(knn_model, newdata = absorpTest)

Get the test Set performance metrics

postResample(pred = knn_pred, obs = yTest)

Neural Network without PCA

Create the grid for the network

nn_grid <- expand.grid(.decay = c(0, 0.01, 0.1), .size = 1:10)

Set the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/nnet_model_q3.rds”)) {

```
nnet_model <- readRDS("models/nnet_model_q3.rds")
```

} else {

```
nnet_model <- train(x = absorpTrain, y = yTrain, tuneGrid = nn_grid, method = "nnet
", preProc = c("center", "scale"),

                    linout = TRUE, trace = FALSE, MaxNWts = 10 * (ncol(absorpTrain
)+1) + 10 + 1, maxit=500)

saveRDS(nnet_model, "models/nnet_model_q3.rds")
```

}

Print the model

nnet_model

Plot the model

plot(nnet_model)

Predict the test set

nnet_pred <- predict(nnet_model, newdata = absorpTest)

Get the test Set performance metrics

postResample(pred = nnet_pred, obs = yTest)

Neural Network with PCA

Create the grid for the network

nn_grid <- expand.grid(.decay = c(0, 0.01, 0.1), .size = 1:10)

Set the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/nnet_pca_model_q3.rds”)) {

```
nnet_pca_model <- readRDS("models/nnet_pca_model_q3.rds")
```

} else {

```
nnet_pca_model <- train(x = absorpTrain, y = yTrain, tuneGrid = nn_grid, method = "
nnet", preProc = c("center", "scale", "pca"),

                    linout = TRUE, trace = FALSE, MaxNWts = 10 * (ncol(absorpTrain
)+1) + 10 + 1, maxit=500)

saveRDS(nnet_pca_model, "models/nnet_pca_model_q3.rds")
```

}

Print the model

nnet_pca_model

Plot the model

plot(nnet_pca_model)

Predict the test set

nnet_pca_pred <- predict(nnet_pca_model, newdata = absorpTest)

Get the test Set performance metrics

postResample(pred = nnet_pca_pred, obs = yTest)

Averaged Neural Network

Create the tune grid

tune_grid <- expand.grid(.decay = c(0, 0.01, .1), .size = 1:10, .bag = FALSE)

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/avg_nnet_model_q3.rds”)) {

```
avg_nnet_model <- readRDS("models/avg_nnet_model_q3.rds")
```

} else {

```
avg_nnet_model <- train(x = absorpTrain, y = yTrain, tuneGrid = tune_grid, method =
"avNNet", preProc = c("center", "scale"),

                        linout = TRUE, trace = FALSE, MaxNWts = 10 * (ncol(absorpT
rain) + 1) + 10 + 1, maxit = 500)

saveRDS(avg_nnet_model, "models/avg_nnet_model_q3.rds")
```

}

Print the model

avg_nnet_model

Plot the model

plot(avg_nnet_model)

Make the prediction

avg_nnet_pred <- predict(avg_nnet_model, newdata = absorpTest)

Get the performance scores

postResample(pred = avg_nnet_pred, obs = yTest)

Mars Model with no preprocessing

Create the tune grid

tune_grid <- expand.grid(.degree = 1:3, .nprune = 2:38)

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/mars_model_q3.rds”)) {

```
mars_model <- readRDS("models/mars_model_q3.rds")
```

} else {

```
mars_model <- train(x = absorpTrain, y = yTrain, trControl = ctrl, tuneGrid = tune_
grid, method = "earth")

saveRDS(mars_model, "models/mars_model_q3.rds")
```

}

Print the model

mars_model

Plot the model

plot(mars_model)

Make the prediction

mars_pred <- predict(mars_model, newdata = absorpTest)

Get the performance scores

postResample(pred = mars_pred, obs = yTest)

Mars Model with spatial sign removing correlated predictors

Remove the highly correlated value

highlyCorDescr <- findCorrelation(absorp_df, cutoff = 0.999)

absorp_cor_df <- absorp_df[,-highlyCorDescr]

Creating the test and train after removing correlated predictors

absorpCorTrain <- as.data.frame(absorp_cor_df[cv_index])

absorpCorTest <- as.data.frame(absorp_cor_df[-cv_index])

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/mars_model_Spatial_q3.rds”)) {

```
mars_model <- readRDS("models/mars_model_Spatial_q3.rds")
```

} else {

```
mars_model <- train(x = absorpTrain, y = yTrain, trControl = ctrl, preProcess = "sp
atialSign", tuneGrid = tune_grid, method = "earth")

saveRDS(mars_model, "models/mars_model_Spatial_q3.rds")
```

}

Print the model

mars_model

Plot the model

plot(mars_model)

Make the prediction

mars_pred <- predict(mars_model, newdata = absorpTest)

Get the performance scores

postResample(pred = mars_pred, obs = yTest)

Support Vector Machine

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/svm_model_q3.rds”)) {

```
svm_model <- readRDS("models/svm_model_q3.rds")
```

} else {

```
svm_model <- train(x = absorpTrain, y = yTrain, trControl = ctrl, tuneLength = 14,
method = "svmRadial", preProc = c("center","scale"))

saveRDS(svm_model, "models/svm_model_q3.rds")
```

}

Print the model

svm_model

Plot the model

plot(svm_model)

Make the prediction

svm_pred <- predict(svm_model, newdata = absorpTest)

Get the performance scores

postResample(pred = svm_pred, obs = yTest)

Question 4

data(permeability)

fingerprints_df <- as.data.frame(fingerprints)

head(fingerprints_df[, 1:5])

permeability[1:5]

fingerprints_filtered_df <- fingerprints_df[, -nearZeroVar(fingerprints_df)]

print(paste("Number of predictors left out for modeling is", dim(fingerprints_filtered_df)[2]))

Setting the seed for reproduciablity

set.seed(1)

Performing data spliting

cv_index <- createDataPartition(permeability, p = 0.8, list = FALSE)

fingerprintsTrain <- fingerprints_filtered_df[cv_index,]

fingerprintsTest <- fingerprints_filtered_df[-cv_index,]

permeabilityTrain <- permeability[cv_index]

permeabilityTest <- permeability[-cv_index]

Setting up the control parameter

ctrl <- trainControl(method = "LGOCV")

KNN

Set the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the KNN model

if(file.exists("models/knn_model_q4.rds")) {

```
knn_model <- readRDS("models/knn_model_q4.rds")
```

} else {

```
knn_model <- train(x = fingerprintsTrain, y = permeabilityTrain, method = "knn", pr
eProc = c("center", "scale"), trControl = ctrl, tuneLength = 10)

saveRDS(knn_model, "models/knn_model_q4.rds")
```

}

Print the model

knn_model

Plot the model

plot(knn_model)

Predict the model

knn_pred <- predict(knn_model, newdata = fingerprintsTest)

Get the test Set performance metrics

postResample(pred = knn_pred, obs = permeabilityTest)

Neural Network

Create the grid for the network

nn_grid <- expand.grid(.decay = c(0, 0.01, 0.1), .size = 1:10)

Set the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/nnet_model_q4.rds”)) {

```
nnet_model <- readRDS("models/nnet_model_q4.rds")
```

} else {

```
nnet_model <- train(x = fingerprintsTrain, y = permeabilityTrain, tuneGrid = nn_gri
d, method = "nnet", preProc = c("center", "scale"), trControl = ctrl,

                    linout = TRUE, trace = FALSE, MaxNWts = 10 * (ncol(fingerprint
sTrain)+1) + 10 + 1, maxit=500)

saveRDS(nnet_model, "models/nnet_model_q4.rds")
```

}

Print the model

nnet_model

Plot the model

plot(nnet_model)

Predict the test set

nnet_pred <- predict(nnet_model, newdata = fingerprintsTest)

Get the test Set performance metrics

postResample(pred = nnet_pred, obs = permeabilityTest)

Mars Model with no preprocessing

Create the tune grid

tune_grid <- expand.grid(.degree = 1:3, .nprune = 2:38)

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/mars_model_q4.rds”)) {

```
mars_model <- readRDS("models/mars_model_q4.rds")
```

} else {

```
mars_model <- train(x = fingerprintsTrain, y = permeabilityTrain, trControl = ctrl,
tuneGrid = tune_grid, method = "earth")

saveRDS(mars_model, "models/mars_model_q4.rds")
```

}

Print the model

mars_model

Plot the model

plot(mars_model)

Make the prediction

mars_pred <- predict(mars_model, newdata = fingerprintsTest)

Get the performance scores

postResample(pred = mars_pred, obs = permeabilityTest)

Mars Model removing correlation

Remove the highly correlated value

highlyCorDescr <- findCorrelation(fingerprints_filtered_df, cutoff = .75)

fingerprints_filtered_cor_df <- fingerprints_filtered_df[,-highlyCorDescr]

Split the data

fingerprintsCorTrain <- fingerprints_filtered_cor_df[cv_index,]

fingerprintsCorTest <- fingerprints_filtered_cor_df[-cv_index,]

Create the tune grid

tune_grid <- expand.grid(.degree = 1:3, .nprune = 2:38)

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/mars_model_cor_q4.rds”)) {

```
mars_model <- readRDS("models/mars_model_cor_q4.rds")
```

} else {

```
mars_model <- train(x = fingerprintsCorTrain, y = permeabilityTrain, trControl = ct
rl, preProcess = "spatialSign", tuneGrid = tune_grid, method = "earth")

saveRDS(mars_model, "models/mars_model_cor_q4.rds")
```

}

Print the model

mars_model

Plot the model

plot(mars_model)

Make the prediction

mars_pred <- predict(mars_model, newdata = fingerprintsCorTest)

Get the performance scores

postResample(pred = mars_pred, obs = permeabilityTest)

Support Vector Machine

Setting the seed

set.seed(1)

Check the file exists and load to variables

else bulid and store the model

if(file.exists(“models/svm_model_q4.rds”)) {

```
svm_model <- readRDS("models/svm_model_q4.rds")
```

} else {

```
svm_model <- train(x = fingerprintsTrain, y = permeabilityTrain, tuneLength = 14, m
ethod = "svmRadial",

                   trControl = ctrl, preProc = c("center","scale"))

saveRDS(svm_model, "models/svm_model_q4.rds")
```

}

Print the model

svm_model

Plot the model

```
plot(svm_model)

# Make the prediction

svm_pred <- predict(svm_model, newdata = fingerprintsTest)

# Get the performance scores

postResample(pred = svm_pred, obs = permeabilityTest)
```

End of the Assignemnt