Homework Assignment for Chapter 7

7.1 Simulate a single predictor and a nonlinear relationship.

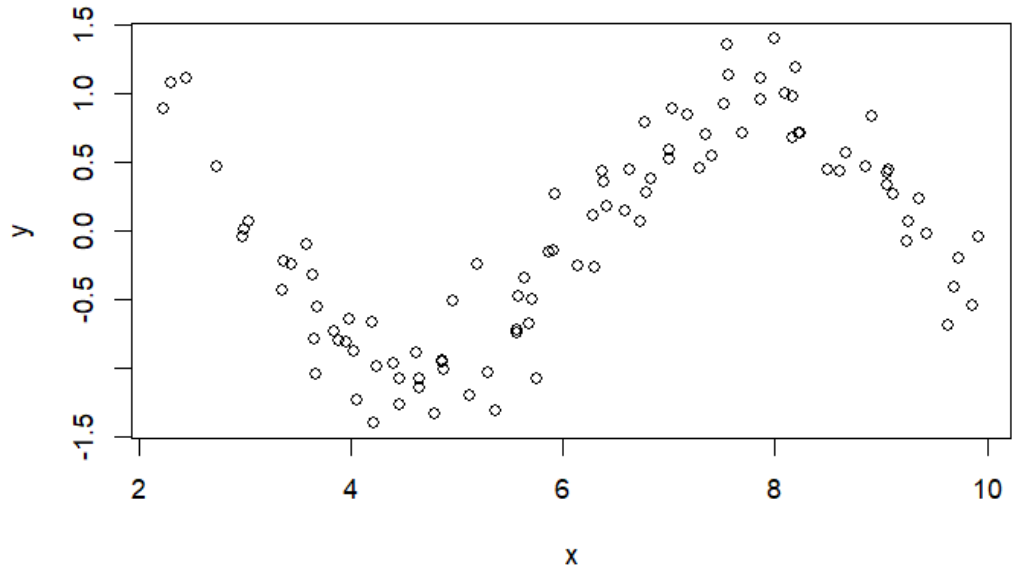

Fig 1: Plot of x vs y simulated values

a) Fit different models using a radial basis function.

The graphs below show the sine data. Putting sigma as automatic in SVM kernel, the svm-rbf model gives the results as shown based on different values of cost and epsilon.

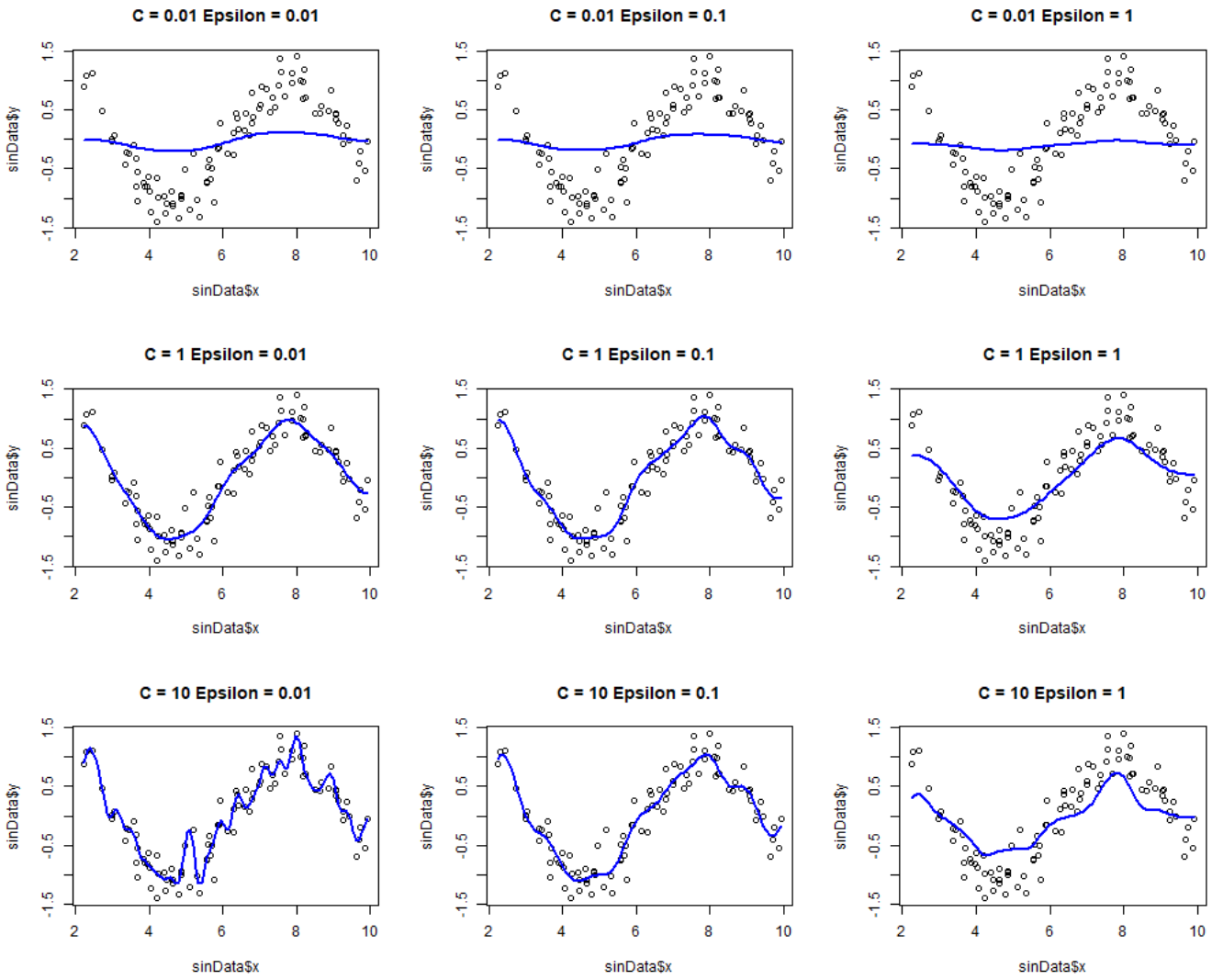

Fig 2: Different values of cost and epsilon impacting the model fit

From the graphs, the model becomes more jagged with the increase in cost. Higher cost values result to overfitting and lower cost values results to underfitting. The impact of the epsilon is also similar but in the opposite direction. As the epsilon decreases, the model fits overfits more and underfits when epsilon increases. The effect is however smaller compared to the cost effect. Therefore, if the value of the epsilon is decreased and the value of cost is increased, then the model becomes overfitted. This is similar in relation to increasing the value of epsilon and

decreasing the value of cost as its underfits the model. From the graph, the optimal model had **cost(C)=1** and **epsilon=0.1**

b)How do the cost , epsilon and sigma values affect the model.

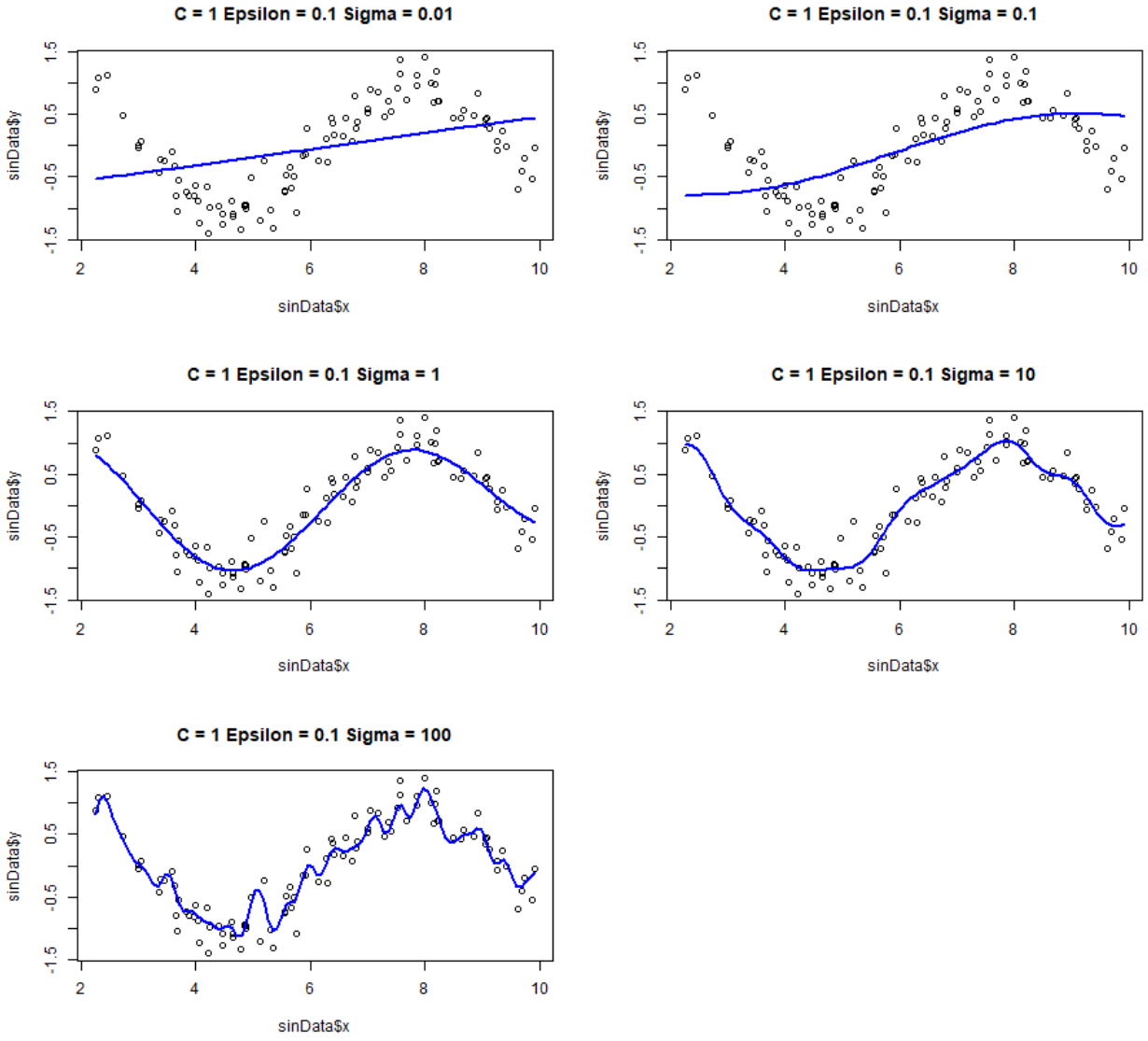

Figure 3: Different values of sigma impacting the optimal model fit

From Fig 2, it was evident that cost and epsilon impact the model fit. Higher cost value and lower value for epsilon overfits while a lower value for cost and higher value for epsilon underfits. Fig 3 shows the fitted regression curves. A smaller value for sigma results in underfitting while a larger value results in overfitting. The relationship between sigma, cost and epsilon is that higher value for cost and sigma and lower value for epsilon overfits the data while lower value for cost and sigma and higher value for epsilon overfits the data. The optimal value for sigma from the graph was **sigma=1**.

7.2 Friedman(1991)

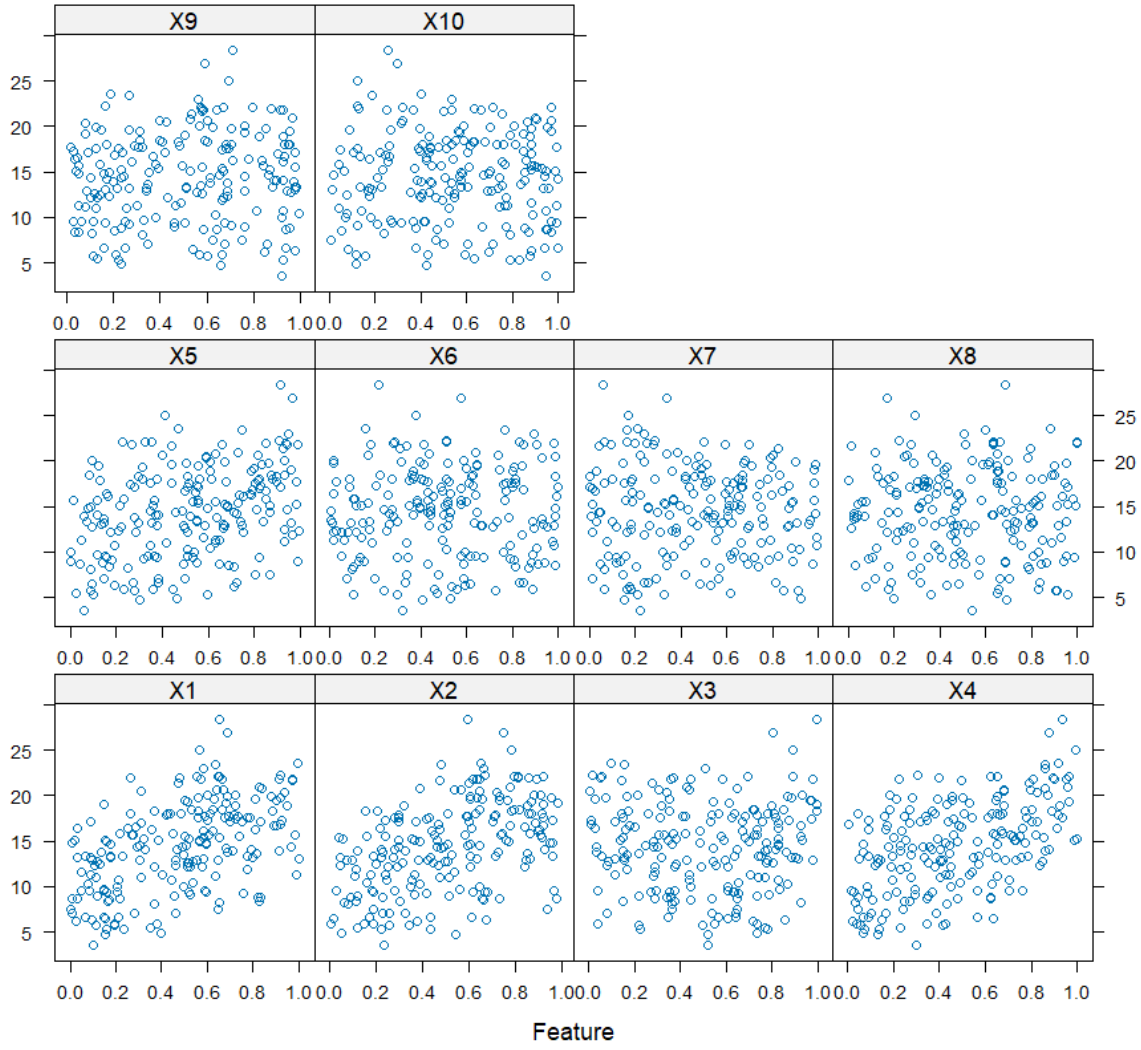

Fig 4: Feature plots

Tune several models on these data

a) **KNN and MARS**

KNN Model

```
> knnModel
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  3.466085  0.5121775  2.816838
   7  3.349428  0.5452823  2.727410
   9  3.264276  0.5785990  2.660026
  11  3.214216  0.6024244  2.603767
  13  3.196510  0.6176570  2.591935
  15  3.184173  0.6305506  2.577482
  17  3.183130  0.6425367  2.567787
  19  3.198752  0.6483184  2.592683
  21  3.188993  0.6611428  2.588787
  23  3.200458  0.6638353  2.604529

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 17.
```

```
> postResample(pred = knnPred, obs = testData$y)
     RMSE  Rsquared       MAE
3.2040595 0.6819919 2.5683461
```

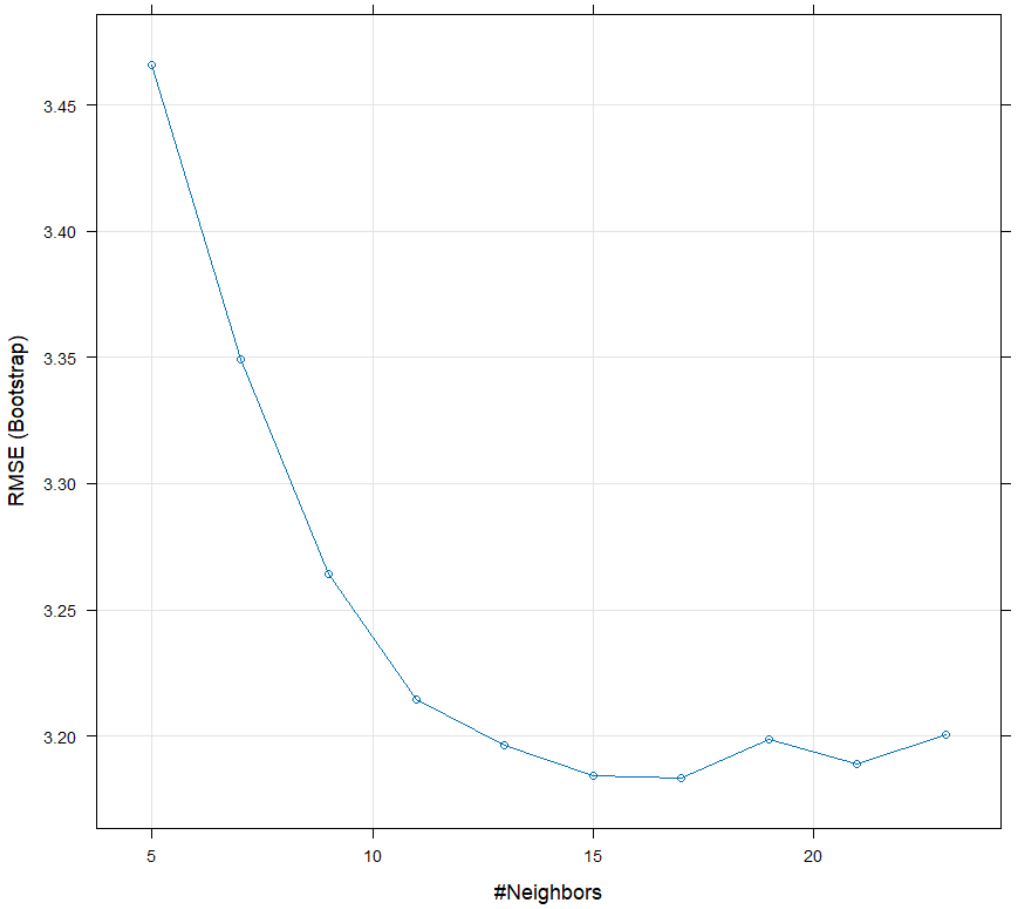

Fig 5:Plot of RMSE vs Tuning parameter(number of Neighbors) from the KNN model.

MARS Model

```
Multivariate Adaptive Regression Spline

200 samples
 10 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
Resampling results across tuning parameters:
```

degree	nprune	RMSE	Rsquared	MAE
1	2	4.327937	0.2544880	3.600474
1	3	3.572450	0.4912720	2.895811
1	4	2.596841	0.7183600	2.106341
1	5	2.370161	0.7659777	1.918669
1	6	2.276141	0.7881481	1.810001
1	7	1.766728	0.8751831	1.390215
1	8	1.780946	0.8723243	1.401345
1	9	1.665091	0.8819775	1.325515
1	10	1.663804	0.8821283	1.327657
1	11	1.657738	0.8822967	1.331730
1	12	1.653784	0.8827903	1.331504
1	13	1.648496	0.8823663	1.316407
1	14	1.639073	0.8841742	1.312833
1	15	1.639073	0.8841742	1.312833
1	16	1.639073	0.8841742	1.312833
1	17	1.639073	0.8841742	1.312833
1	18	1.639073	0.8841742	1.312833
1	19	1.639073	0.8841742	1.312833
1	20	1.639073	0.8841742	1.312833
1	21	1.639073	0.8841742	1.312833
1	22	1.639073	0.8841742	1.312833
1	23	1.639073	0.8841742	1.312833
1	24	1.639073	0.8841742	1.312833
1	25	1.639073	0.8841742	1.312833
1	26	1.639073	0.8841742	1.312833
1	27	1.639073	0.8841742	1.312833
1	28	1.639073	0.8841742	1.312833
1	29	1.639073	0.8841742	1.312833
1	30	1.639073	0.8841742	1.312833
1	31	1.639073	0.8841742	1.312833

```
  2         9      1.603578  0.8938666  1.261361
  2        10      1.492421  0.9084998  1.168700
  2        11      1.317350  0.9292504  1.033926
  2        12      1.304327  0.9320133  1.019108
  2        13      1.277510  0.9323681  1.002927
  2        14      1.269626  0.9350024  1.003346
  2        15      1.266217  0.9359400  1.013893
  2        16      1.268470  0.9354868  1.011414
  2        17      1.268470  0.9354868  1.011414
  2        18      1.268470  0.9354868  1.011414
  2        19      1.268470  0.9354868  1.011414
  2        20      1.268470  0.9354868  1.011414
  2        21      1.268470  0.9354868  1.011414
  2        22      1.268470  0.9354868  1.011414
  2        23      1.268470  0.9354868  1.011414
  2        24      1.268470  0.9354868  1.011414
  2        25      1.268470  0.9354868  1.011414
  2        26      1.268470  0.9354868  1.011414
  2        27      1.268470  0.9354868  1.011414
  2        28      1.268470  0.9354868  1.011414
  2        29      1.268470  0.9354868  1.011414
  2        30      1.268470  0.9354868  1.011414
  2        31      1.268470  0.9354868  1.011414
  2        32      1.268470  0.9354868  1.011414
  2        33      1.268470  0.9354868  1.011414
  2        34      1.268470  0.9354868  1.011414
  2        35      1.268470  0.9354868  1.011414
  2        36      1.268470  0.9354868  1.011414
  2        37      1.268470  0.9354868  1.011414
  2        38      1.268470  0.9354868  1.011414
  2        39      1.268470  0.9354868  1.011414
  2        40      1.268470  0.9354868  1.011414
  2        41      1.268470  0.9354868  1.011414
  2        42      1.268470  0.9354868  1.011414
  2        43      1.268470  0.9354868  1.011414
  2        44      1.268470  0.9354868  1.011414
  2        45      1.268470  0.9354868  1.011414
  2        46      1.268470  0.9354868  1.011414
  2        47      1.268470  0.9354868  1.011414
  2        48      1.268470  0.9354868  1.011414
  2        49      1.268470  0.9354868  1.011414
  2        50      1.268470  0.9354868  1.011414

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were nprune = 15 and degree = 2.

> postResample(marsPred, testData$y)
     RMSE  Rsquared       MAE
1.1589948 0.9460418 0.9250230
```

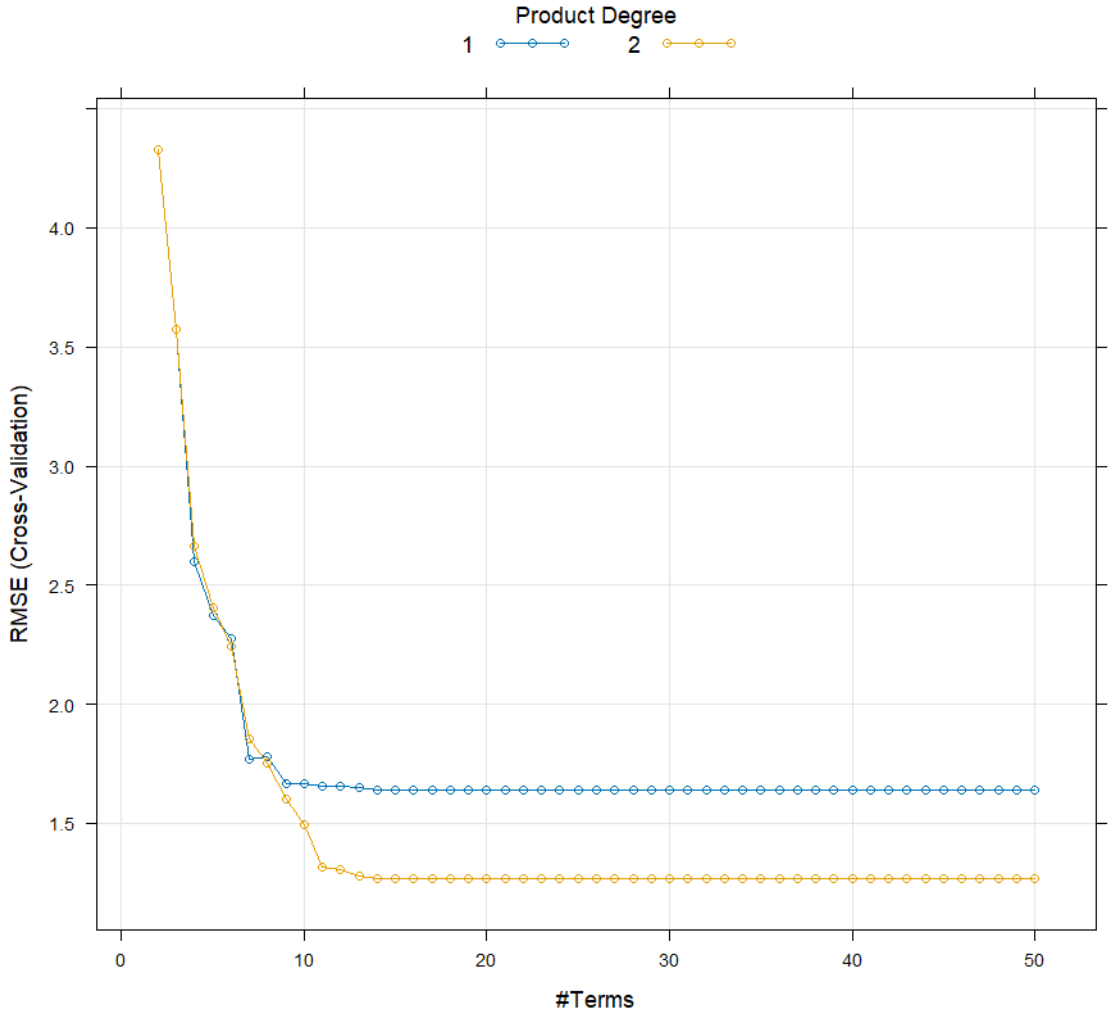

Fig 6: Plot of RMSE vs Tuning parameters(number of Terms) for product degree 1 & 2

Model	Best Tuning Parameter	Training		Testing	
		RMSE	R^2	RMSE	R^2
KNN	K=17	3.18313	0.642537	3.20406	0.681992
MARS	nprune=15, degree=2	1.266217	0.93594	1.1589948	0.9460418

Table 1: Summary of KNN and MARS Models performance

Table 1 gives a summary of model performance for KNN and MARS models. The best tuning parameter for KNN model was 17 nearest neighbors. This parameter returned an RMSE of 3.183 and Rsquared of 64.25% on training data and an RMSE of 3.20406 and Rsquared of 68.199% on testing data.

In the MARS model, the best tuning parameters were nprune=15(number of terms) and degree=2. These returned an RMSE of 1.2662 and Rsquared of 93.59% on training and RMSE of 1.15899 and Rsquared of 94.604%. The RMSE increased in testing compared to training for KNN model while reduced in the MARS model. From the results, it is evident that MARS model was the best performing model with a lower RMSE and higher Rsquared on both testing and training data.

b) Informative predictors

MARS model was the best performing model from the results with the lowest RMSE and highest Rsquared.

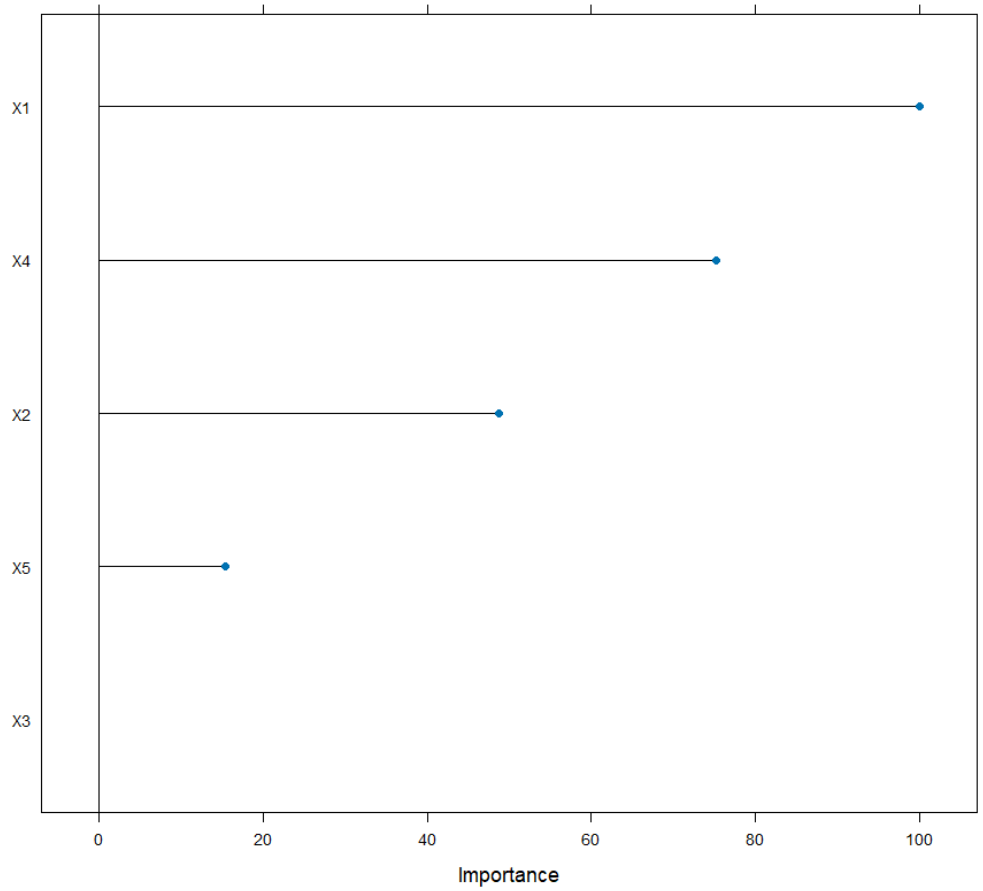

Fig 6: Informative predictors from MARS model

From Fig 6, MARS selects the informative variables, X1-X5, however, one variable (X3) is not included as it seems to be an insignificant variable. It had an importance of 0. The most important variable was X1, X4, X2, X5 respectively.

7.5) Chemical Manufacturing process

a) Optimal resampling and test set performance

KNN Model

```
k-Nearest Neighbors

144 samples
 46 predictor

Pre-processing: centered (46), scaled (46)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 144, 144, 144, 144, 144, 144, ...
Resampling results across tuning parameters:

  k   RMSE       Rsquared   MAE
   1  0.8069868  0.3585479  0.6221810
   2  0.7945381  0.3452558  0.6215228
   3  0.7798144  0.3557838  0.6160841
   4  0.7727716  0.3623802  0.6219222
   5  0.7724599  0.3616489  0.6242203
   6  0.7711153  0.3630012  0.6228637
   7  0.7706809  0.3636719  0.6267207
   8  0.7732782  0.3589050  0.6318418
   9  0.7753867  0.3580938  0.6364829
  10  0.7786901  0.3558245  0.6393581
  11  0.7791005  0.3577678  0.6401123
  12  0.7788577  0.3595527  0.6389717
  13  0.7780303  0.3640244  0.6374259
  14  0.7809504  0.3592364  0.6390881
  15  0.7811662  0.3622560  0.6399824
  16  0.7805569  0.3665142  0.6398642
  17  0.7803479  0.3695689  0.6405658
  18  0.7820266  0.3700396  0.6436663
  19  0.7840892  0.3687509  0.6451415
  20  0.7831851  0.3729102  0.6429685

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 7.
```

```
> postResample(pred = knnPred, test_y)
     RMSE  Rsquared       MAE
0.7753100 0.6291244 0.5650785
```

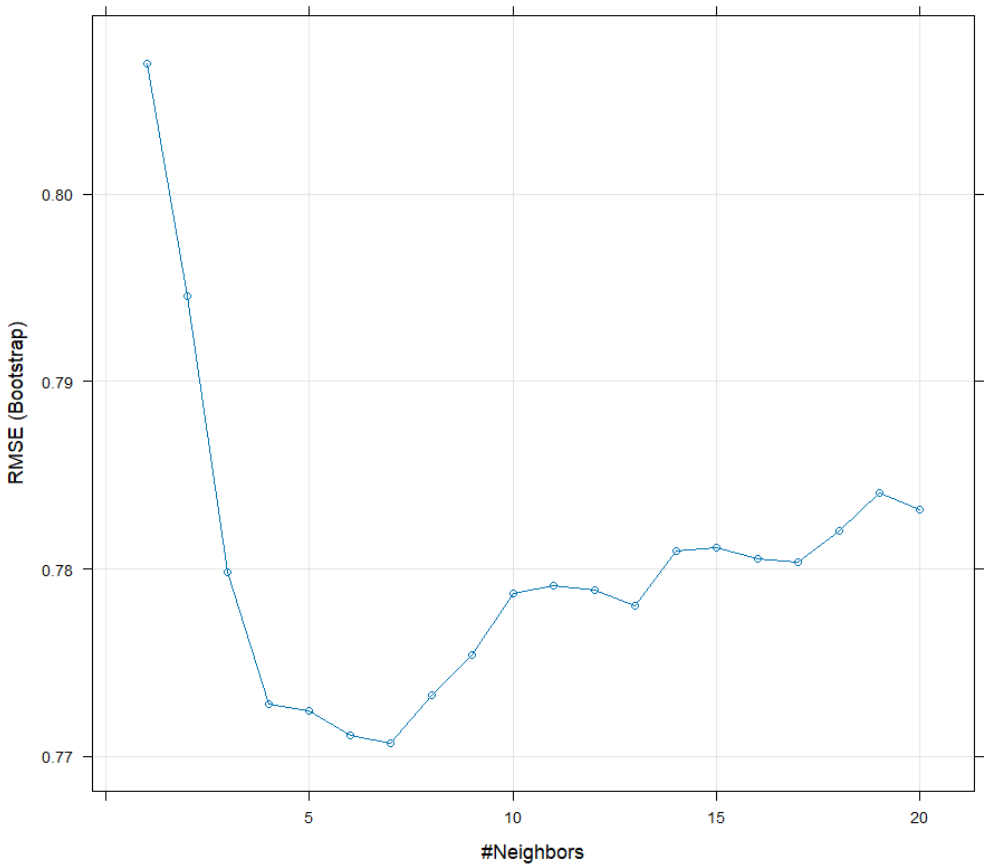

Fig 7: Plot of RMSE vs Tuning parameters(Number of Neighbors) for KNN model

MARS Model

```
Multivariate Adaptive Regression Spline

144 samples
 46 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 129, 130, 130, 130, 130, 130, ...
Resampling results across tuning parameters:

  degree  nprune  RMSE       Rsquared   MAE
  1        2      0.7550068  0.4307265  0.6102915
  1        3      0.6933356  0.5017883  0.5596249
  1        4      0.6900458  0.5185380  0.5475502
  1        5      0.6783332  0.5262967  0.5357266
  1        6      0.6675478  0.5480722  0.5296198
  1        7      0.6715504  0.5474077  0.5365508
  1        8      0.6723747  0.5540396  0.5309450
  1        9      0.6743078  0.5579412  0.5290980
  1       10      0.9310468  0.5340670  0.6180044
  1       11      0.8776743  0.5461891  0.6096892
  1       12      0.8885013  0.5434908  0.6263087
  1       13      0.8849684  0.5559341  0.6239571
  1       14      0.8846432  0.5596580  0.6244523
  1       15      0.8977908  0.5525943  0.6388835
  1       16      0.9057133  0.5393514  0.6469981
  1       17      0.9057133  0.5393514  0.6469981
  1       18      0.8347345  0.5506506  0.6329036
  1       19      0.8462507  0.5478437  0.6443194
  1       20      0.8462507  0.5478437  0.6443194
  1       21      0.8462507  0.5478437  0.6443194
  1       22      0.8462507  0.5478437  0.6443194
  1       23      0.8462507  0.5478437  0.6443194
  1       24      0.8835469  0.5482033  0.6577434
  1       25      0.8947031  0.5477458  0.6646712
  1       26      0.8826346  0.5486522  0.6558955
  1       27      0.8823737  0.5479984  0.6589410
  1       28      0.8844366  0.5483556  0.6589241
  1       29      0.8746605  0.5493141  0.6554806
  1       30      0.8832411  0.5470469  0.6650363
  1       31      0.8832411  0.5470469  0.6650363
  1       32      0.8832411  0.5470469  0.6650363
  1       33      0.8832411  0.5470469  0.6650363
  1       34      0.8832411  0.5470469  0.6650363
```

```
2         9      0.7248030  0.4922085  0.5794311
2        10      0.7417045  0.4934168  0.5783342
2        11      0.7561229  0.4906913  0.5809195
2        12      0.7555818  0.4953327  0.5791017
2        13      0.7602874  0.5082639  0.5773860
2        14      0.7941251  0.4840313  0.6007022
2        15      0.7930741  0.4817035  0.5996386
2        16      0.8019851  0.4846175  0.6077373
2        17      0.8026834  0.4830225  0.6114521
2        18      0.7996637  0.4914152  0.6083964
2        19      0.7978614  0.4952797  0.6060352
2        20      0.7995833  0.5049545  0.6025830
2        21      0.8145037  0.4876630  0.6148706
2        22      0.8174762  0.4887815  0.6174026
2        23      0.8207979  0.4863844  0.6206596
2        24      0.8207979  0.4863844  0.6206596
2        25      0.8207979  0.4863844  0.6206596
2        26      0.8207979  0.4863844  0.6206596
2        27      0.8207979  0.4863844  0.6206596
2        28      0.8207979  0.4863844  0.6206596
2        29      0.8207979  0.4863844  0.6206596
2        30      0.8207979  0.4863844  0.6206596
2        31      0.8207979  0.4863844  0.6206596
2        32      0.8207979  0.4863844  0.6206596
2        33      0.8207979  0.4863844  0.6206596
2        34      0.8207979  0.4863844  0.6206596
2        35      0.8207979  0.4863844  0.6206596
2        36      0.8207979  0.4863844  0.6206596
2        37      0.8207979  0.4863844  0.6206596
2        38      0.8207979  0.4863844  0.6206596
2        39      0.8207979  0.4863844  0.6206596
2        40      0.8207979  0.4863844  0.6206596
2        41      0.8207979  0.4863844  0.6206596
2        42      0.8207979  0.4863844  0.6206596
2        43      0.8207979  0.4863844  0.6206596
2        44      0.8207979  0.4863844  0.6206596
2        45      0.8207979  0.4863844  0.6206596
2        46      0.8207979  0.4863844  0.6206596
2        47      0.8207979  0.4863844  0.6206596
2        48      0.8207979  0.4863844  0.6206596
2        49      0.8207979  0.4863844  0.6206596
2        50      0.8207979  0.4863844  0.6206596

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were nprune = 6 and degree = 1.

> postResample(marsPred, test_y)
     RMSE  Rsquared       MAE
0.6072243 0.7326667 0.4554900
```

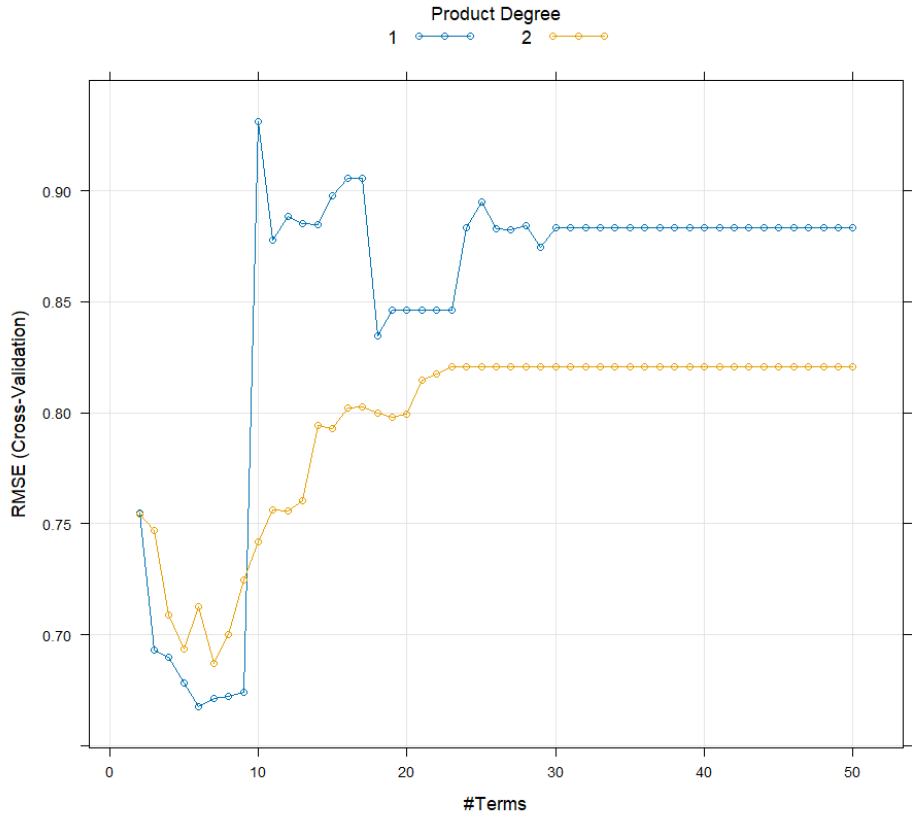

Fig 8: Plot of RMSE vs Tuning parameter(Number of terms) for product degree 1 and 2 in MARS model.

SVM with RBF Model

```
Support Vector Machines with Radial Basis Function Kernel

144 samples
 46 predictor

Pre-processing: centered (46), scaled (46)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 129, 130, 130, 130, 130, 130, ...
Resampling results across tuning parameters:

  C        RMSE       Rsquared   MAE
     0.25  0.7429565  0.4750802  0.6203319
     0.50  0.6916184  0.5124496  0.5732121
     1.00  0.6486353  0.5610596  0.5361512
     2.00  0.6291883  0.5854112  0.5174082
     4.00  0.6208814  0.5997014  0.5107844
     8.00  0.6236907  0.6018006  0.5146059
    16.00  0.6232154  0.6015026  0.5160291
    32.00  0.6232154  0.6015026  0.5160291
    64.00  0.6232154  0.6015026  0.5160291
   128.00  0.6232154  0.6015026  0.5160291
   256.00  0.6232154  0.6015026  0.5160291
   512.00  0.6232154  0.6015026  0.5160291
  1024.00  0.6232154  0.6015026  0.5160291
  2048.00  0.6232154  0.6015026  0.5160291

Tuning parameter 'sigma' was held constant at a value of 0.01500334
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.01500334 and C = 4.
```

```
> postResample(svmRPred, test_y)
     RMSE  Rsquared       MAE
0.5782027 0.7918000 0.4054779
```

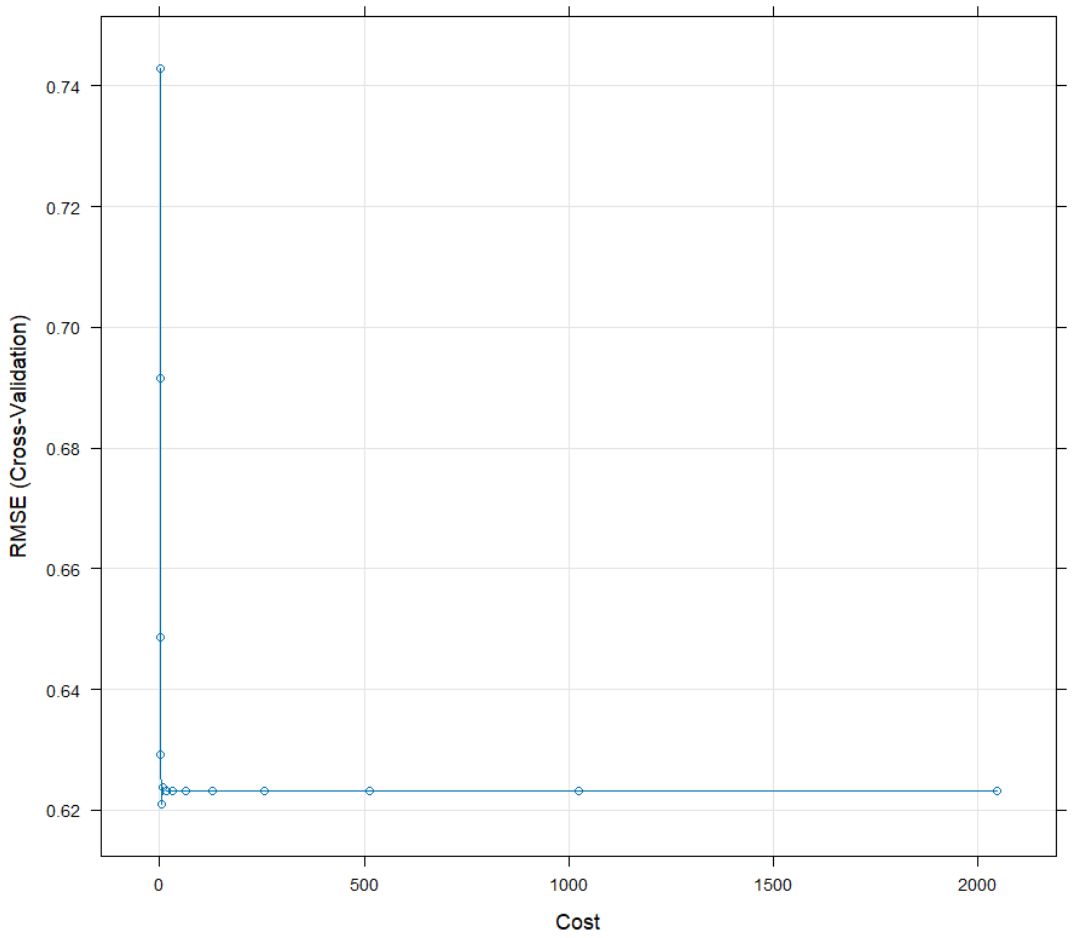

Fig 9: Plot of RMSE vs Tuning parameter(Cost) for SVM model with Radial Basis Function as kernel

Neural Network

```
> inherits
Model Averaged Neural Network

144 samples
 46 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 128, 129, 130, 130, 130, 130, ...
Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared   MAE
  0.001   1    0.7653179  0.4399746  0.6131414
  0.001   2    0.7659032  0.4858164  0.6160634
  0.001   3    0.7576686  0.5322251  0.5975380
  0.001   4    0.7978923  0.4855311  0.6398387
  0.001   5    0.7434336  0.5178471  0.5926740
  0.001   6    0.7104251  0.5437339  0.5637122
  0.001   7    0.6928611  0.5873595  0.5481558
  0.001   8    0.6889780  0.5665006  0.5415493
  0.001   9    0.7103682  0.5768907  0.5641110
  0.001  10    0.7072856  0.5605502  0.5516822
  0.010   1    0.7826676  0.4245697  0.6185172
  0.010   2    0.7372593  0.5074392  0.5862820
  0.010   3    0.7983248  0.5031797  0.6065042
  0.010   4    0.7323901  0.5268294  0.5901470
  0.010   5    0.6914327  0.5673163  0.5496812
  0.010   6    0.6579770  0.6099522  0.5234227
  0.010   7    0.7051084  0.5674190  0.5673728
  0.010   8    0.7281623  0.5410291  0.5794148
  0.010   9    0.7188286  0.5459944  0.5604357
  0.010  10    0.7278590  0.5456443  0.5727689
  0.100   1    0.7505041  0.5011754  0.6014165
  0.100   2    0.7274221  0.5558847  0.5667350
  0.100   3    0.7308408  0.5513994  0.5807114
  0.100   4    0.7157271  0.5375045  0.5548671
  0.100   5    0.7594482  0.5078457  0.5907204
  0.100   6    0.6981229  0.5738428  0.5372227
  0.100   7    0.7156359  0.5629324  0.5686512
  0.100   8    0.7091180  0.5644682  0.5655018
  0.100   9    0.7370128  0.5325374  0.5842544
  0.100  10    0.7266033  0.5306193  0.5746917
  1.000   1    0.6582854  0.5677369  0.5343741
  1.000   2    0.6609648  0.5719522  0.5201934
  1.000   3    0.6321444  0.6113486  0.5009317

  0.100   2    0.7274221  0.5558847  0.5667350
  0.100   3    0.7308408  0.5513994  0.5807114
  0.100   4    0.7157271  0.5375045  0.5548671
  0.100   5    0.7594482  0.5078457  0.5907204
  0.100   6    0.6981229  0.5738428  0.5372227
  0.100   7    0.7156359  0.5629324  0.5686512
  0.100   8    0.7091180  0.5644682  0.5655018
  0.100   9    0.7370128  0.5325374  0.5842544
  0.100  10    0.7266033  0.5306193  0.5746917
  1.000   1    0.6582854  0.5677369  0.5343741
  1.000   2    0.6609648  0.5719522  0.5201934
  1.000   3    0.6321444  0.6113486  0.5009317
  1.000   4    0.6410163  0.6036537  0.5034400
  1.000   5    0.6696718  0.5739582  0.5260560
  1.000   6    0.6588407  0.5869738  0.5216078
  1.000   7    0.6736898  0.5729621  0.5325310
  1.000   8    0.6762450  0.5664972  0.5313624
  1.000   9    0.6758233  0.5677196  0.5338653
  1.000  10    0.6708570  0.5745623  0.5338642

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 = 1 and bag = FALSE.
```

```
> postResample(nnPred, test_y)
     RMSE  Rsquared       MAE
0.5684723 0.7546658 0.4351397
```

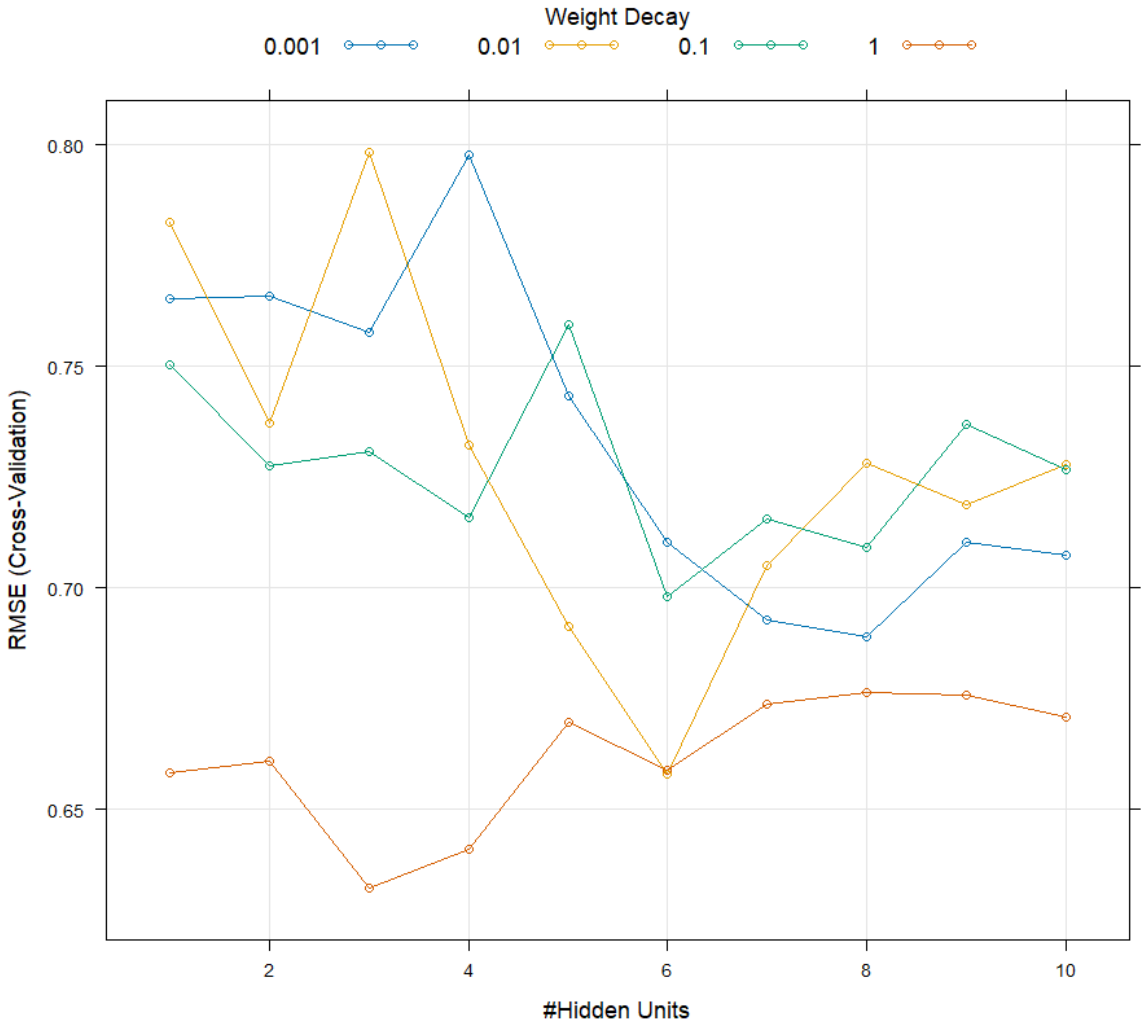

Fig 10: Plot of RMSE vs Hidden Units for Neural Network model

Model	Best Tuning Parameter	Training		Testing	
		RMSE	R^2	RMSE	R^2
KNN	K=7	0.7706809	0.3636719	0.7753100	0.6291244
MARS	nprune=6, degree=1	0.6675478	0.5480722	0.6072243	0.7326667
SVMR	C=4, sigma=0.01500334	0.6208814	0.59997014	0.5782027	0.791800
Neural Network	Size(Hidden Units)=3, decay=1	0.6321444	0.6113486	0.5684723	0.7546658

Table 2: Summary for nonlinear regression models

From Table 2, KNN model was underperforming on the training data. It had the lowest R^2 of 36.36% and the highest RMSE of 0.7706809. Neural network was the best performing model on the training data at 61.135% followed closely by SVMR at 59.997%. However, SVMR had the lowest RMSE of 0.620881 compared to Neural Network with 0.6321444. MARS model had an RMSE of 0.6675 and Rsquared of 54.807% on training data.

On the testing data, SVMR was the best performing model with an R^2 of 79.18% and an RMSE of 0.5782027. KNN was the worst performing model with an R^2 of 62.91% and an RMSE of 0.7753100, which was the highest. The RMSE increases on KNN for testing compared to training while it reduces in the other three models. This proves that the other three models were best performing on the testing set. The R^2 increases on all models for testing with SVMR returning the highest performance. Based on the results on training and testing, SVMR is the best performing model with the highest Rsquared and lowest RMSE on testing.

b) Important predictors

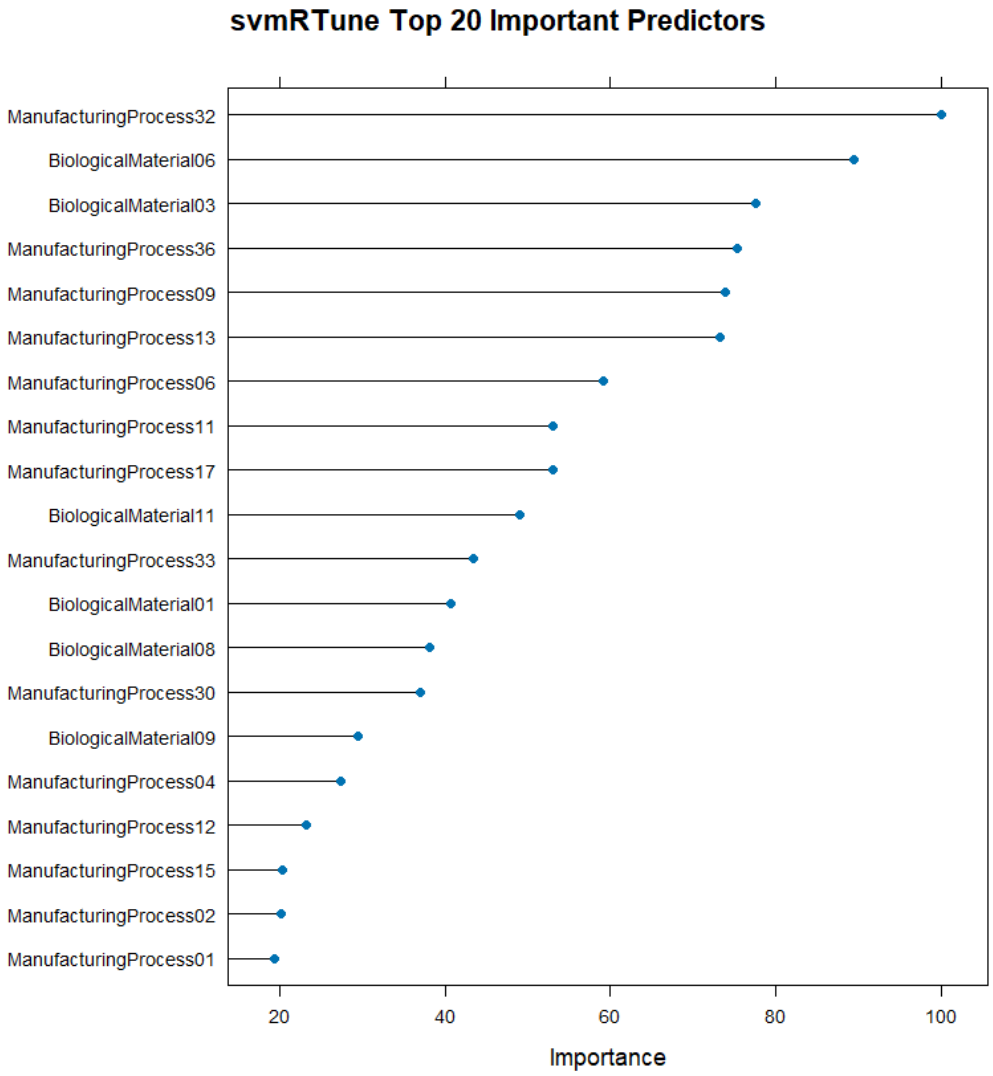

Fig 11: Plot of most important predictors from the optimal model(SVMR model) from the most important to the least important.

Figure 11 plots the top 20 most important predictors in relation to the SVMR model which was the optimal model. ManufacturingProcess 32 was the most important predictor with ManufcaturingProcessing01 being the 20th most important. Process predictors dominated the most important predictors with 14 predictors in the top 20 while Biological had 6 predictors. This shows that Process predictors were the most important predictors in the optimal nonlinear regression model.

The top 5 most important predictors were ManufacturingProcess32, BiologicalMaterial06, BiologicalMaterial03, ManufacturingProcess36, and ManufacturingProcess09 respectively.

c) **Compare**

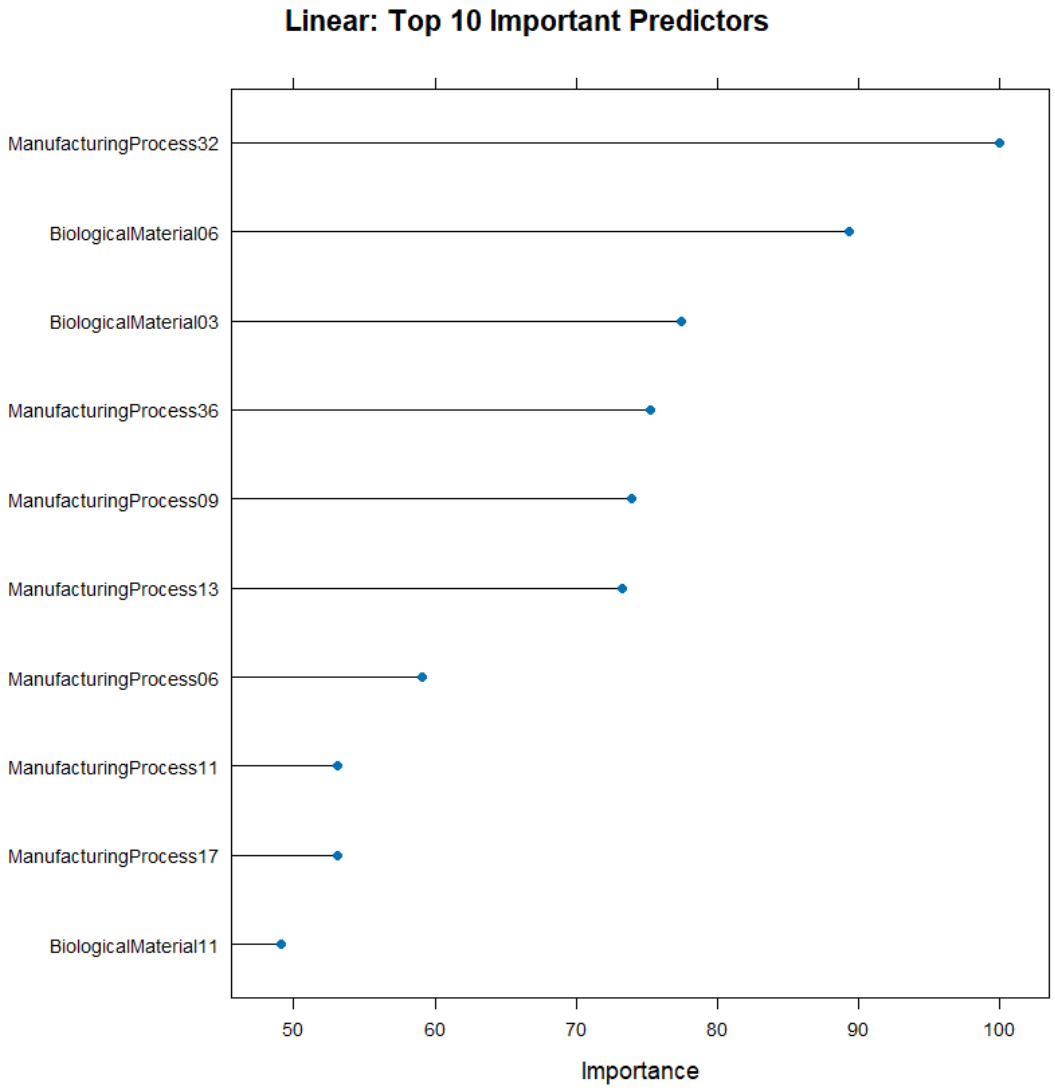

Fig 12: Graph of the most important predictors from optimal linear model(Elastic net)

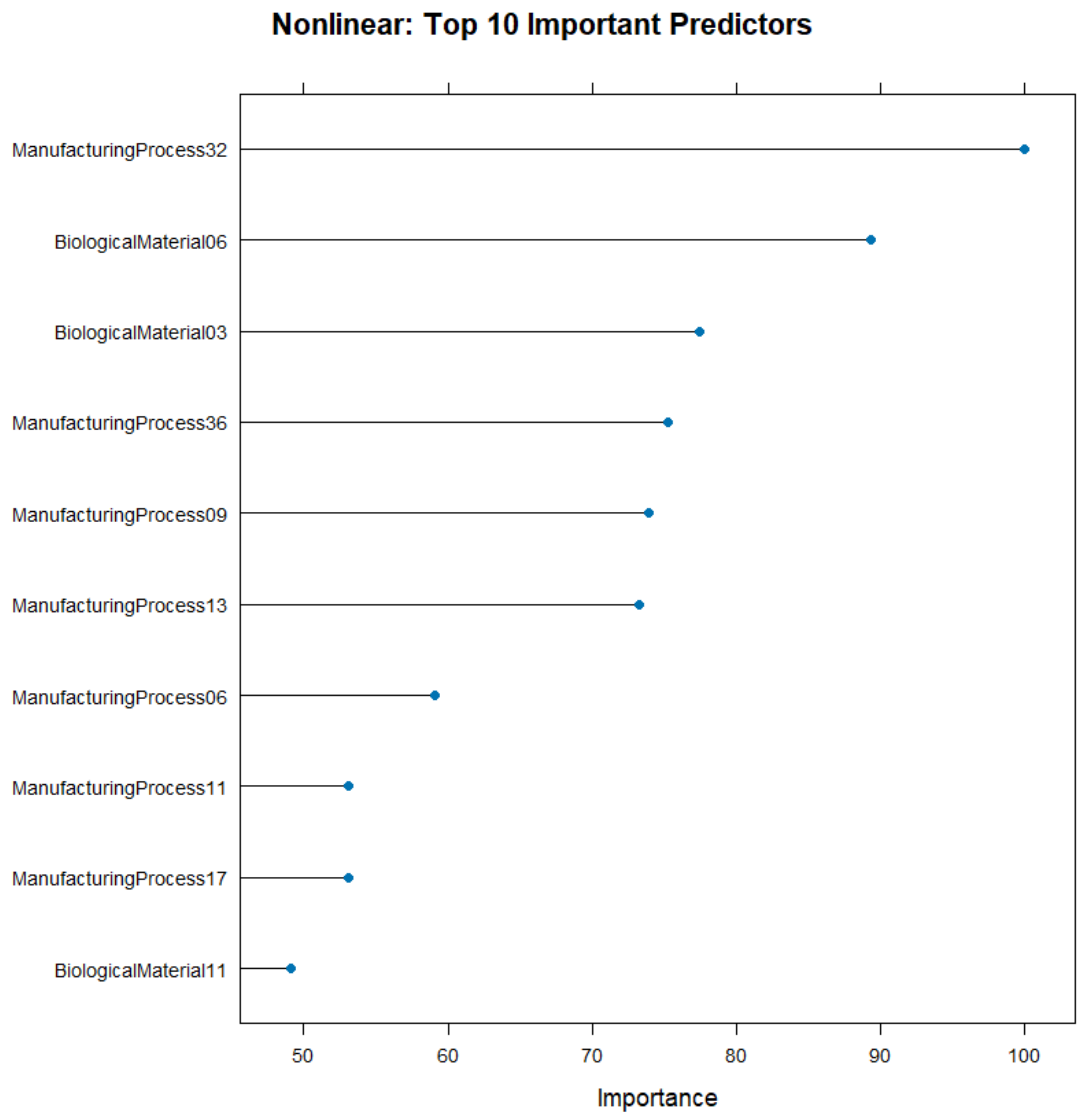

Fig 13: Graph of the most important predictors from the optimal nonlinear model(SVMR)

Type	Predictors	Percent
Process	**7**	**70%**
Biological	**3**	**30%**

The top 10 predictors for both the linear and nonlinear models are the same. No changes have been observed. From Fig 12&13, out of the top 10 predictors, 70% are process variables while 30% are biological variables for both the linear and nonlinear models. The ManufacturingProcess32 is still the topmost predictor for the two different models.

APPENDIX

```
## Question 7.1

set.seed(100)
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))

points(x = dataGrid$x, y = modelPrediction[,1],
       type = "l", col = "blue")
#part a
install.packages("kernlab")
library(kernlab)

# Fit the SVM models with different C and epsilon values
C_values <- c(0.01, 1, 10)
epsilon_values <- c(0.01,0.1,1)



# Set the plotting area to a 3x3 layout
par(mfrow=c(3, 3))

# Loop over the C and epsilon values
```

```
for (C_value in C_values) {
  for (epsilon_value in epsilon_values) {
    # Fit the SVM model
    rbfSVM <- ksvm(x = x, y = y, data = sinData,
                   kernel = "rbfdot", kpar = "automatic",
                   C = C_value, epsilon = epsilon_value)

    # Generate predictions from the model
    modelPrediction <- predict(rbfSVM, newdata = dataGrid)

    # Plot the original data
    plot(sinData$x, sinData$y, main = paste("C =", C_value, "Epsilon =", epsilon_value))

    # Add the model predictions to the plot
    lines(dataGrid$x, modelPrediction, col = "blue", lwd = 2)
  }
}



#part b

# Define a range of sigma values to test

sigma_values <- c(0.01, 0.1, 1, 10,100)
par(mfrow=c(3, 2))

for (sigma_value in sigma_values) {
  rbfSVM <- ksvm(x = x, y = y, data = sinData,
```

```
        kernel = "rbfdot", kpar = list(sigma = sigma_value),
        C = 1, epsilon = 0.1)
  modelPrediction <- predict(rbfSVM, newdata = dataGrid)
  plot(sinData$x, sinData$y, main = paste("C =",1, "Epsilon =",
                                          0.1, "Sigma =", sigma_value))
  lines(dataGrid$x, modelPrediction, col = "blue", lwd = 2)
}


## Question 7.2
library(caret)
library(mlbench)
set.seed(200)
trainingData <- mlbench.friedman1(200, sd = 1)

## We convert the 'x' data from a matrix to a data frame

trainingData$x <- data.frame(trainingData$x)

featurePlot(trainingData$x, trainingData$y)
## or other methods.
## This creates a list with a vector 'y' and a matrix
## of predictors 'x'. Also simulate a large test set to
## estimate the true error rate with good precision:
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)
```

```
## Model Tuning
# Part a

##KNN
library(caret)
knnModel <- train(x = trainingData$x,
                  y = trainingData$y,
                  method = "knn",
                  preProc = c("center", "scale"),
                  tuneLength = 10)
knnModel

knnPred <- predict(knnModel, newdata = testData$x)
## The function 'postResample' can be used to get the test set performance values
postResample(pred = knnPred, obs = testData$y)
plot(knnModel, metric='RMSE')

##MARS
install.packages( "earth")
library(earth)
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:50)
## Change 38 to 50

set.seed(100)

# tune
marsTune <- train(trainingData$x, trainingData$y,
                  method = "earth",
```

```
            tuneGrid = marsGrid,
            trControl = trainControl(method = "cv"))

marsTune
plot(marsTune)
marsPred <- predict(marsTune, testData$x)
postResample(marsPred, testData$y)



#part b
varImp(marsTune)
plot(varImp(marsTune))



##Question 7.5
library(AppliedPredictiveModeling)
data(ChemicalManufacturingProcess)

# imputation
miss <- preProcess(ChemicalManufacturingProcess, method = "knnImpute")
Chemical <- predict(miss, ChemicalManufacturingProcess)

# filtering low frequencies
Chemical <- Chemical[, -nearZeroVar(Chemical)]

set.seed(624)
```

```
# index for training
index <- createDataPartition(Chemical$Yield, p = .8, list = FALSE)

# train
train_x <- Chemical[index, -1]
train_y <- Chemical[index, 1]

# test
test_x <- Chemical[-index, -1]
test_y <- Chemical[-index, 1]

# remove predictors to ensure maximum abs pairwise corr between predictors < 0.75
tooHigh <- findCorrelation(cor(train_x), cutoff = .90)

# removing 21 variables
train_x_nnet <- train_x[, -tooHigh]
test_x_nnet <- test_x[, -tooHigh]

#Part a
##KNN
knnModel <- train(train_x_nnet, train_y,
                  method = "knn",
                  preProc = c("center", "scale"),
                  tuneLength = 10,
                  tuneGrid = data.frame(.k=1:20))

knnModel
knnPred <- predict(knnModel, test_x_nnet)
```

```
## The function 'postResample' can be used to get the test set
## perforamnce values
postResample(pred = knnPred, test_y)
plot(knnModel)

##MARS
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:50)
set.seed(100)
# tune
marsTune <- train(train_x_nnet, train_y,
                  method = "earth",
                  tuneGrid = marsGrid,
                  trControl = trainControl(method = "cv"))

marsTune

marsPred <- predict(marsTune, test_x_nnet)

postResample(marsPred, test_y)
plot(marsTune)

## SVM
set.seed(100)
# tune
svmRTune <- train(train_x_nnet, train_y,
                  method = "svmRadial",
                  preProc = c("center", "scale"),
                  tuneLength = 14,
```

```
                trControl = trainControl(method = "cv"))

svmRTune

svmRPred <- predict(svmRTune, test_x_nnet)

postResample(svmRPred, test_y)
plot(svmRTune)

##NN
nnetGrid <- expand.grid(.decay = c(0.001, 0.01, .1,1),
                        .size = c(1:10),
                        ## The next option is to use bagging (see the
                        ## next chapter) instead of different random
                        ## seeds.
                        .bag = FALSE)

nnetTune2 <- train(train_x_nnet, train_y,
                method = "avNNet",
                tuneGrid = nnetGrid,
                trControl = trainControl(method = "cv"),
                linout = TRUE,
                trace = FALSE,
                MaxNWts = 10 * (ncol(train_x_nnet) + 1) + 10 + 1,
                maxit = 500
)
nnetTune2
nnPred <- predict(nnetTune2, test_x_nnet)
```

```
postResample(nnPred, test_y)
plot (nnetTune2)

rbind(knn = postResample(knnPred, test_y),
      nn = postResample(nnPred, test_y),
      mars = postResample(marsPred, test_y),
      svmR = postResample(svmRPred, test_y))



#part b
varImp(svmRTune)
plot(varImp(svmRTune), top = 20,
    main = "svmRTune Top 20 Important Predictors")
#part c


enetGrid <- expand.grid(.lambda = c(0,0.1, by=.01),
                        .fraction = seq(.05, 1, length = 20))
set.seed(100)
enetTune <- train(x = train_x_nnet,
                  y = train_y,
                  method = "enet",
                  tuneGrid = enetGrid,
                  trControl = ctrl,
                  preProc = c("center", "scale"))
enetTune
plot(enetTune, metric='RMSE')
```

```
prediction4 <- predict(enetTune, newdata = test_x_nnet)

postResample(prediction4,test_y)

plot(enetTune, metric='RMSE')



plot(varImp(svmRTune), top = 10,
    main = "Nonlinear: Top 10 Important Predictors")

plot(varImp(enetTune), top = 10,
    main = "Linear: Top 10 Important Predictors")
```