Michigan Technological University, Computer Science

MA 5790 Combined Section - Predictive Modeling Assignment 4

Anil Silwal

November 9, 2018

Question - 7.1 : Simulate a single predictor and a nonlinear relationship, such as a sin wave shown in Fig. 7.7, and investigate the relationship between the cost, ϵ, and kernel parameters for a support vector machine model:

Solution 7.1(a)
The data here is used from given question, which is sine data. Putting σ as automatic in SVM kernel, the model of svm - radial gives following results based on different values of ϵ and cost.

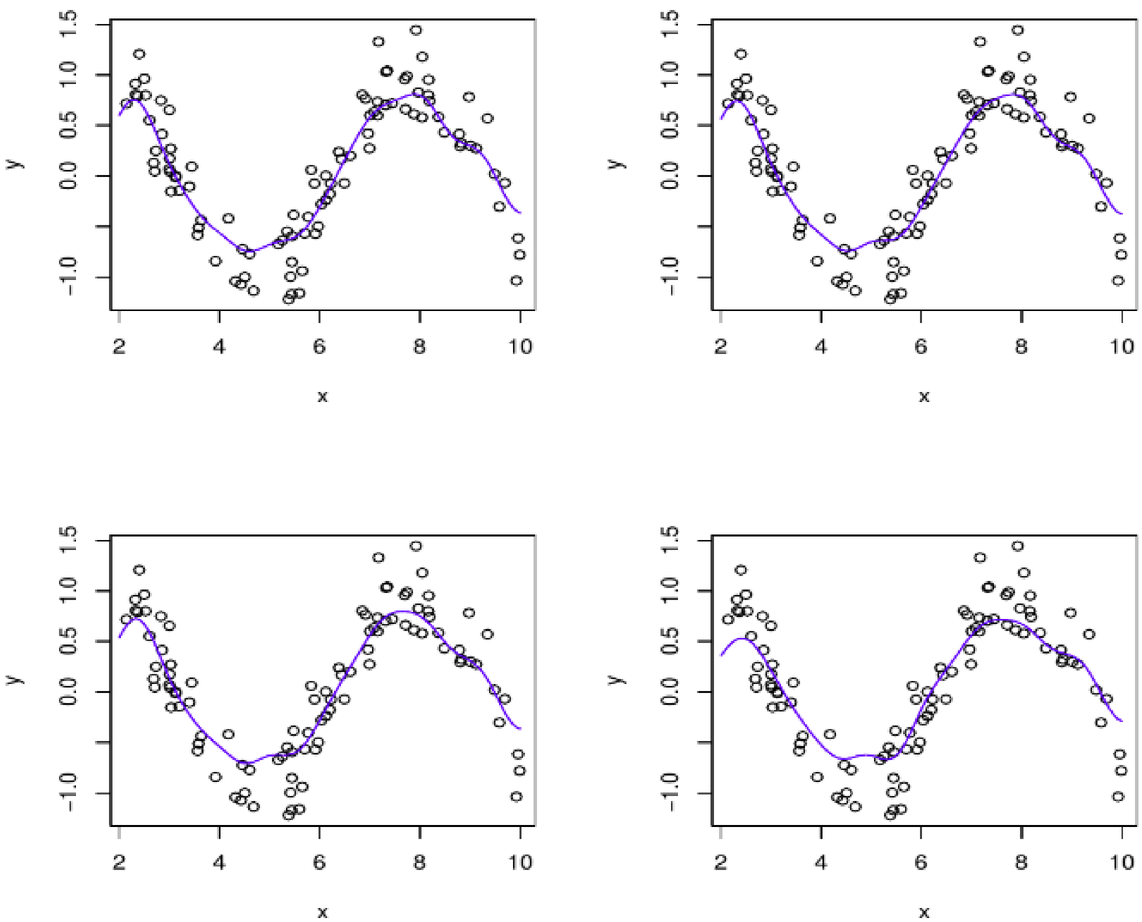

Figure 1: Different values of ϵ and cost that changes the model fit

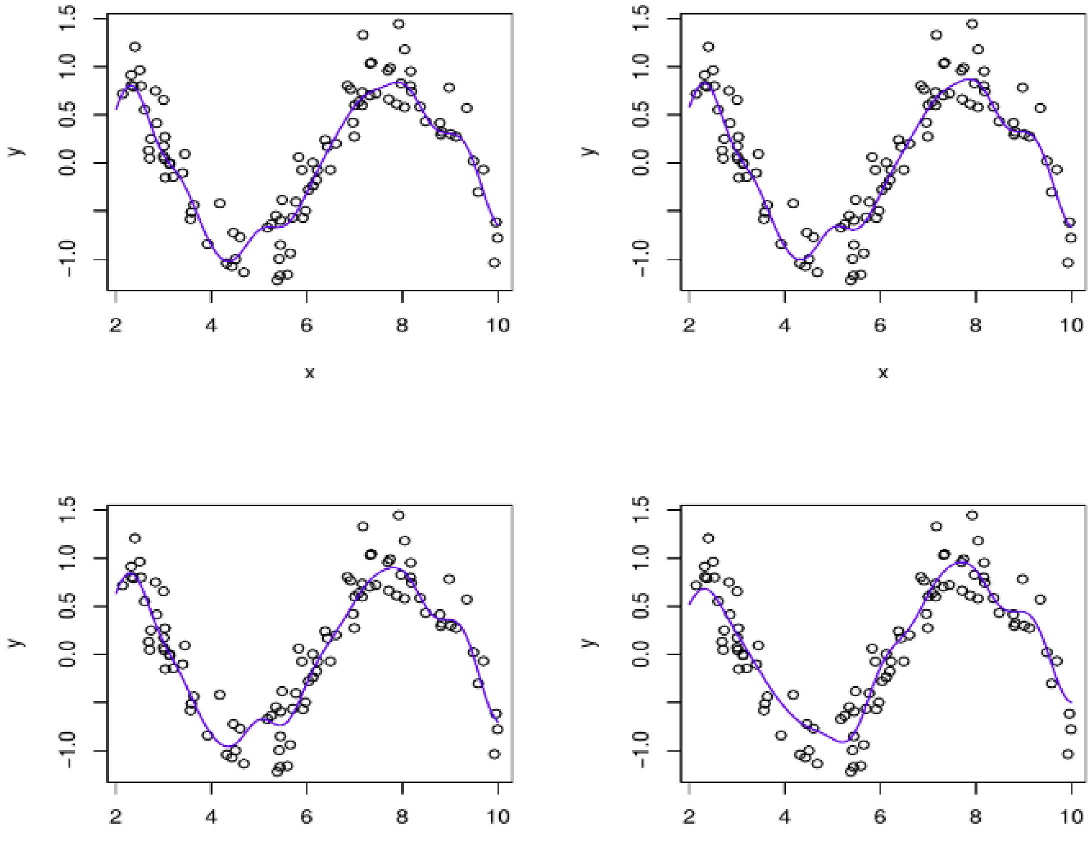

Figure 2: Different values of ϵ and cost that changes the model fit(Contd.)

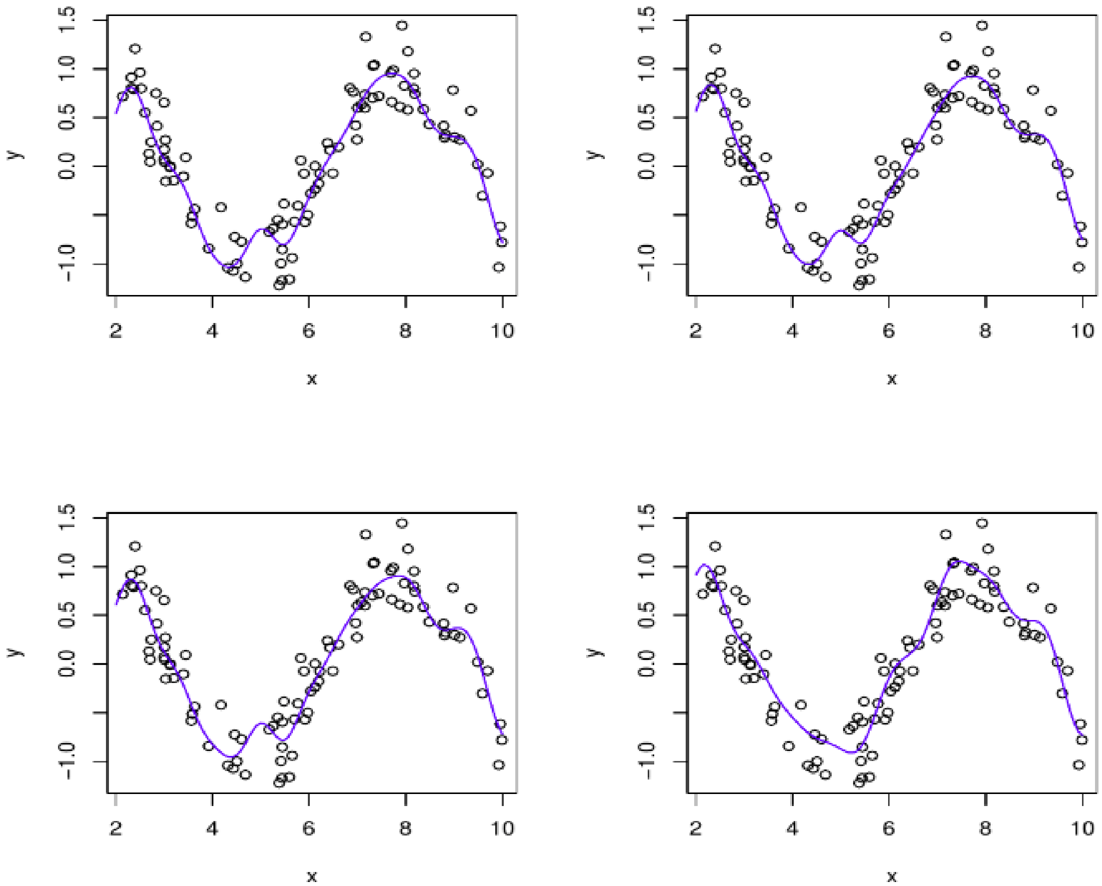

Figure 3: Different values of ϵ and cost that changes the model fit(Contd.)

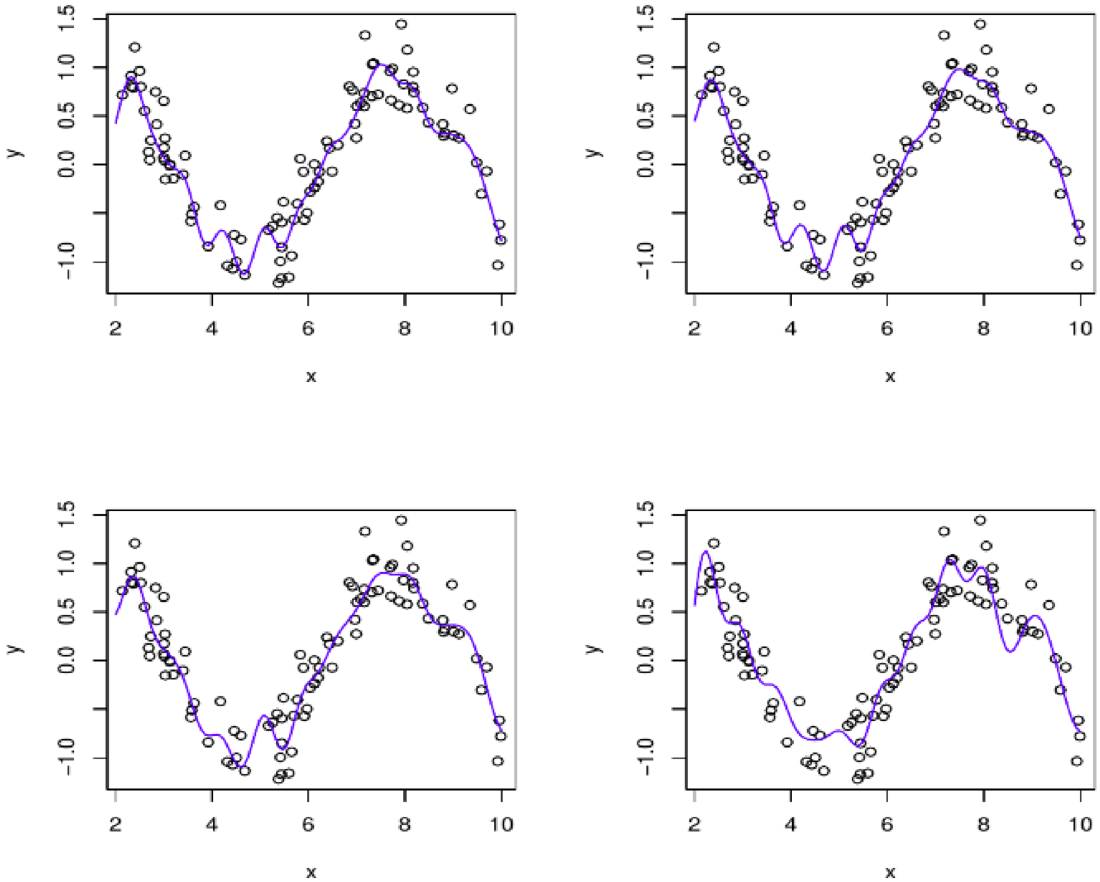

Figure 4: Different values of ϵ and cost that changes the model fit(Contd.)

```
costs    0.25 costs    1.00 costs    4.00 costs 256.00
          400           400           400           400
```

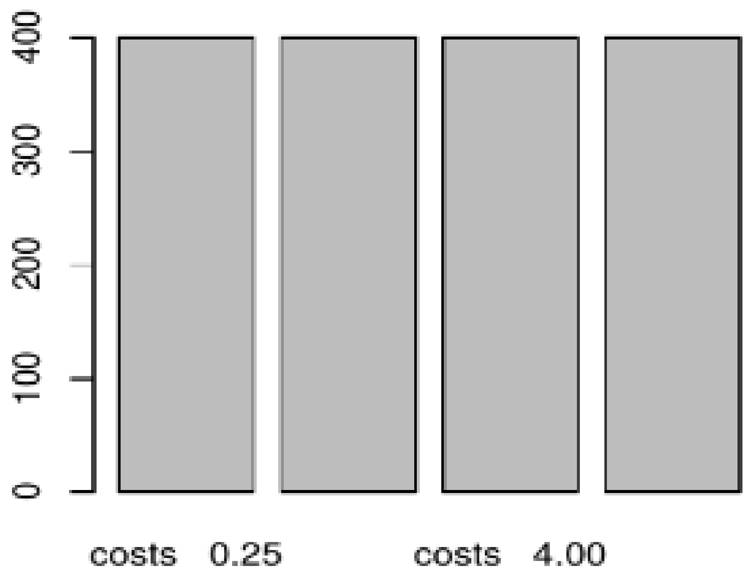

Figure 5: Different values of ϵ and cost that changes the model fit - Summary

Solution 7.1(b)

Instead of putting σ as automatic, giving σ different values as well in SVM kernel, along with different values of ϵ and cost: the model of svm - radial gives following results .

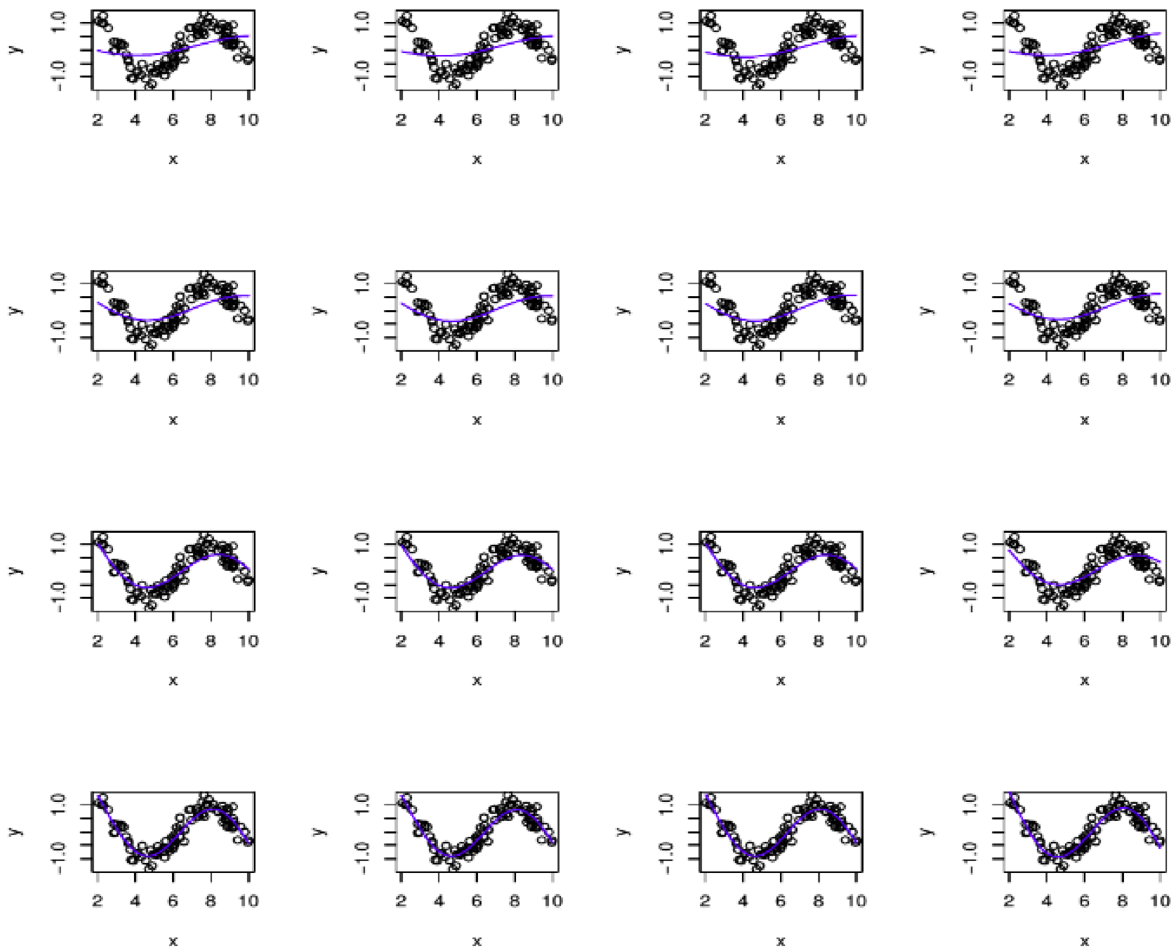

Figure 6: Different values of σ,ϵ and cost that changes the model fit(Contd.)

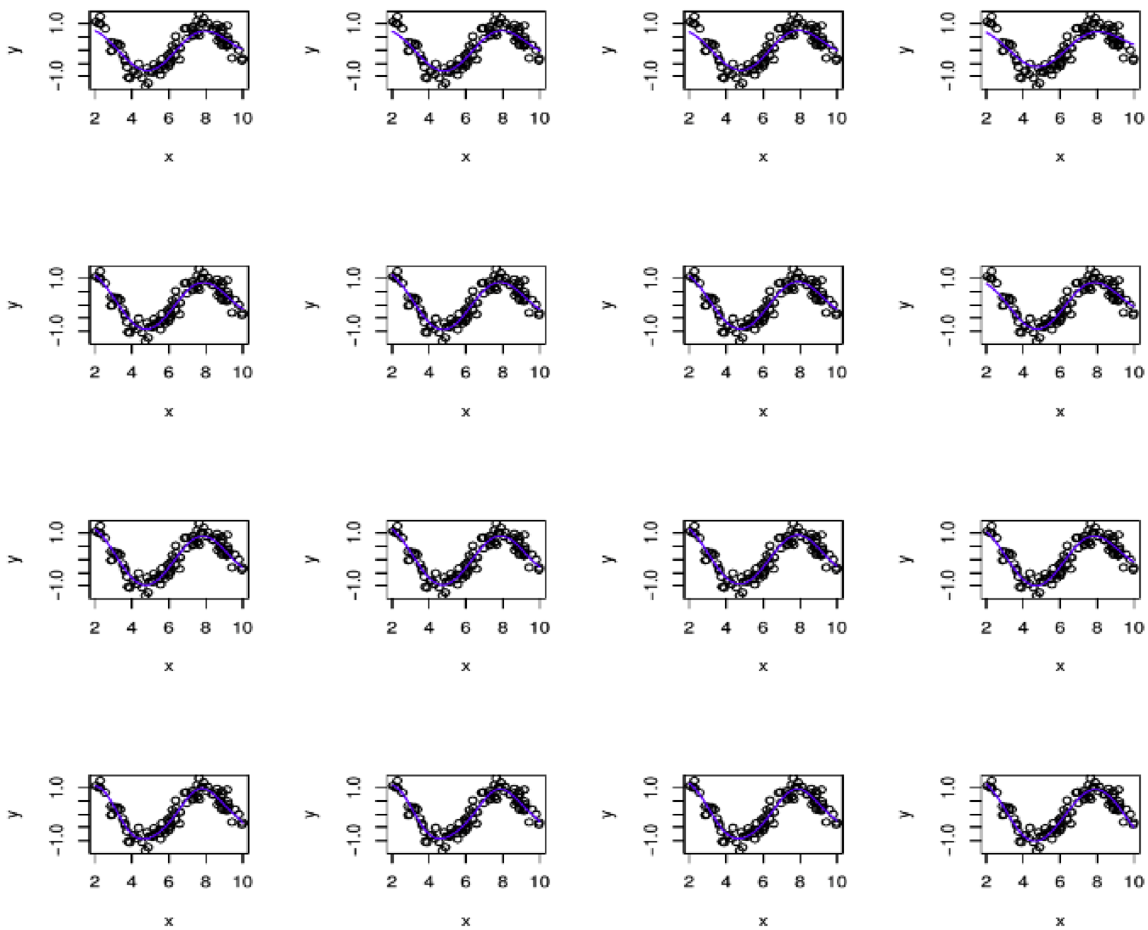

Figure 7: Different values of σ,ϵ and cost that changes the model fit(Contd.)

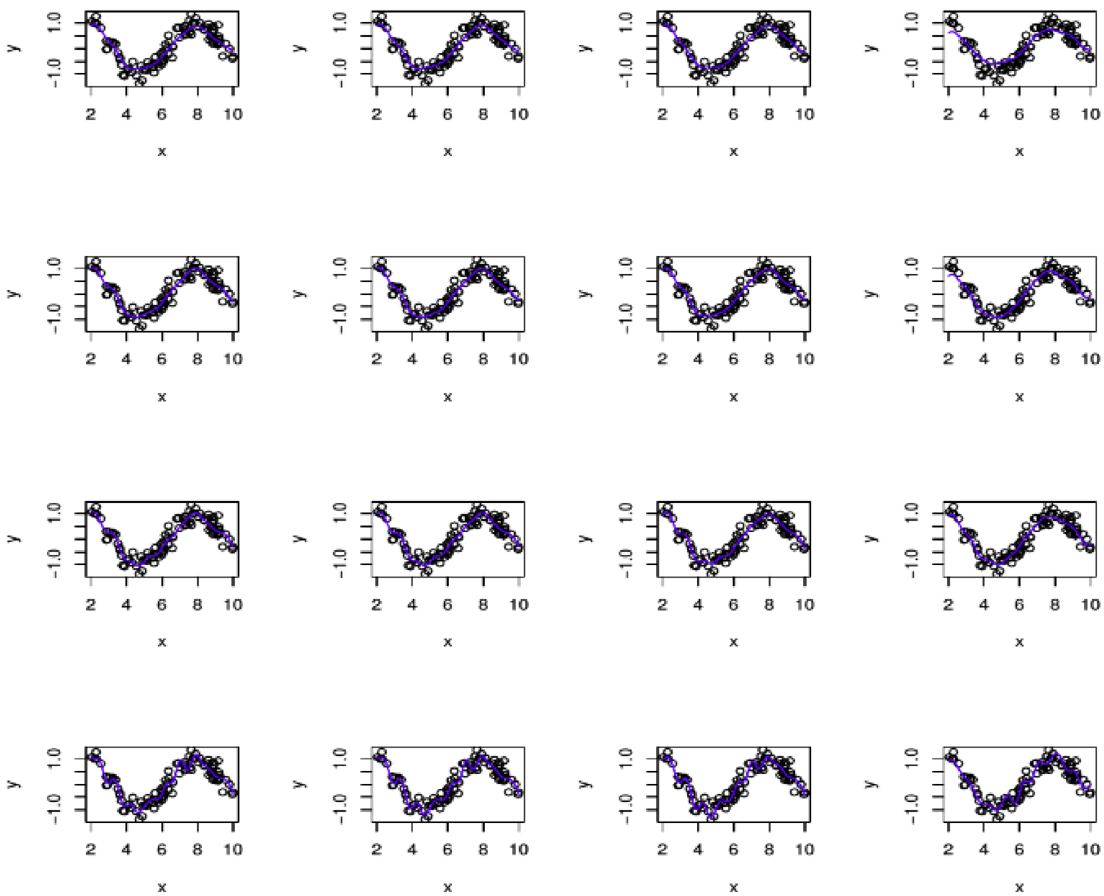

Figure 8: Different values of σ,ϵ and cost that changes the model fit(Contd.)

```
costs   0.25 costs   1.00 costs   4.00 costs 256.00
        1200         1200         1200         1200
sigma  0.19 sigma  0.95 sigma 48.06
       1600        1600        1600
```

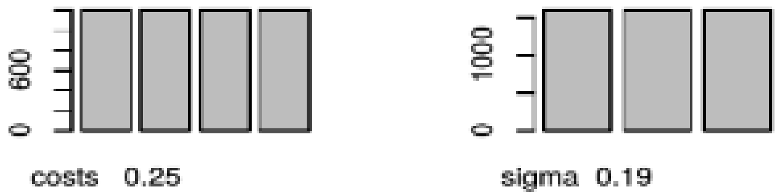

Figure 9: Different values of σ,ϵ and cost that changes the model fit - Summary

Affect of ϵ, σ and cost:
From the results above: for the low to moderate cost values σ appears to effect the model bias. The low cost value tends to underfit the data and the high cost value tends to overfit the SVM-Radial Model.
From Figure: If the value of ϵ is decreased and value of cost is increased, the model is also overfitted.
The figure also shows the powerful fitting ability that SVMs have through tuning parametes. The SVMs are prone to overfitting and should be trained appropriately to protect against the overfitting to the training data.

Question - 7.3 : For the Tecator data described in the last chapter, build SVM, neural network, MARS, and KNN models. Since neural networks are especially sensitive to highly correlated predictors, does pre-processing using PCA help the model?

Solution 7.3:

The MARS, SVM, KNN and Neural Network models based on data tecator are modeled below. The screenshots of each models are given below which itself explains about their result and analysis. Further, a comparative analysis for each model is at last, made to give brief result of each models

```
Call: earth(x=trainAbsorption, y=trainFat)

                coefficients
(Intercept)         95.74671
h(X 8-3.03014)     -89.72102
h(3.40747-X 9)    -133.26088
h(X 9-3.40747)     164.51841
h(3.11181-X 25)    364.94398
h(X 25-3.11181)   -257.36107
h(X 37-3.44309)    177.40212
h(3.43065-X 38)   -266.12270
h(X 38-3.43065)    107.78837
h(X 41-3.70901)    -43.32847
h(X 48-4.01785)    -56.03194
h(4.21139-X 53)     41.42907
h(X 90-3.22475)    -24.13533
h(3.33994-X 98)    -10.70946
h(X 98-3.33994)     13.27460

Selected 15 of 18 terms, and 10 of 100 predictors
Termination condition: RSq changed by less than 0.001 at 18 terms
Importance: X38, X25, X37, X53, X9, X48, X8, X90, X98, X41, X1-unused, X2-unused, ...
Number of terms at each degree of interaction: 1 14 (additive model)
GCV 6.48534    RSS 783.6452    GRSq 0.9594265    RSq 0.9714973
```

Figure 10: MARS using internal GCM for model selection summary

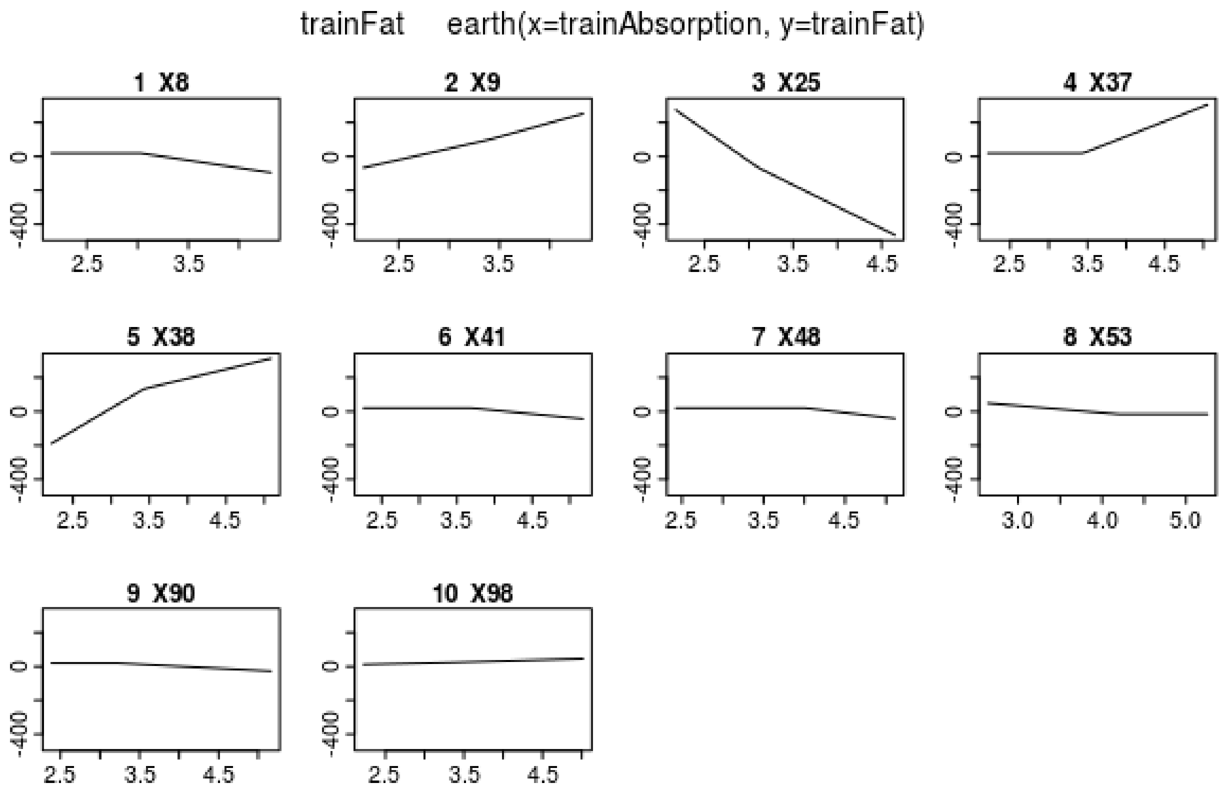

Figure 11: MARS using internal GCM for model selection Plot

```
> marsTuned
Multivariate Adaptive Regression Spline

174 samples
100 predictors

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 4 times)
Summary of sample sizes: 156, 158, 155, 156, 155, 157, ...
Resampling results across tuning parameters:

  degree  nprune  RMSE       Rsquared   MAE
  1        2      10.752420  0.2921011  8.741529
  1        3       9.462329  0.4481313  7.362692
  1        4       8.176167  0.6167719  5.979314
  1        5       4.858111  0.8649621  3.540175
  1        6       4.408066  0.8886124  3.243962
  1        7       3.585074  0.9310057  2.832936
  1        8       3.466494  0.9371832  2.664878
  1        9       3.186672  0.9485908  2.447610
  1       10       2.995577  0.9557192  2.229317
  1       11       2.951124  0.9557693  2.176832
  1       12       2.974570  0.9545542  2.168600
  1       13       2.962047  0.9537802  2.170481
  1       14       2.964096  0.9507689  2.162902
  1       15       2.895931  0.9535763  2.098206
  1       16       2.926419  0.9515805  2.112709
  1       17       2.948424  0.9503899  2.112855
  1       18       2.925744  0.9513087  2.108305
  2        2      10.740251  0.2913762  8.749822
  2        3       9.893063  0.4016583  7.814227
  2        4       8.206524  0.6093851  5.962105
  2        5       4.935943  0.8571862  3.654883
  2        6       4.544491  0.8790799  3.400915
  2        7       3.698305  0.9257263  2.885636
  2        8       3.517154  0.9332658  2.673516
  2        9       3.206127  0.9448427  2.472939
  2       10       3.084692  0.9469320  2.257492

  2       11       3.083035  0.9445038  2.211378
  2       12       3.066316  0.9419510  2.180029
  2       13       2.960571  0.9440065  2.130443
  2       14       2.889224  0.9449148  2.053681
  2       15       2.846749  0.9463640  2.021438
  2       16       2.795918  0.9487813  1.960924
  2       17       2.681071  0.9525823  1.882676
  2       18       2.709793  0.9531480  1.879077

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were nprune = 17 and degree = 2.
```

Figure 12: MARS Tuned Model Summary Using Train()

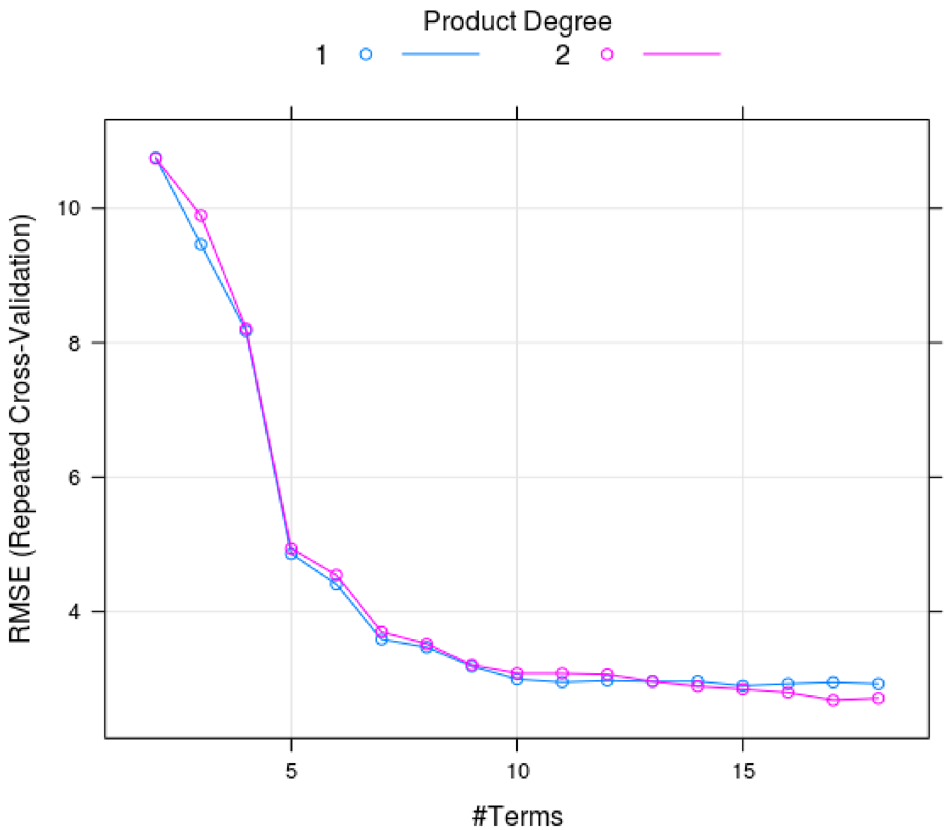

Figure 13: MARS Tuned Model Plot Using Train()

```
> defaultSummary(accuracy)
     RMSE  Rsquared       MAE
2.6474251 0.9603677 1.8821931
```

Figure 14: MARS Accuracy Summary: Prediction - Observation Tuned

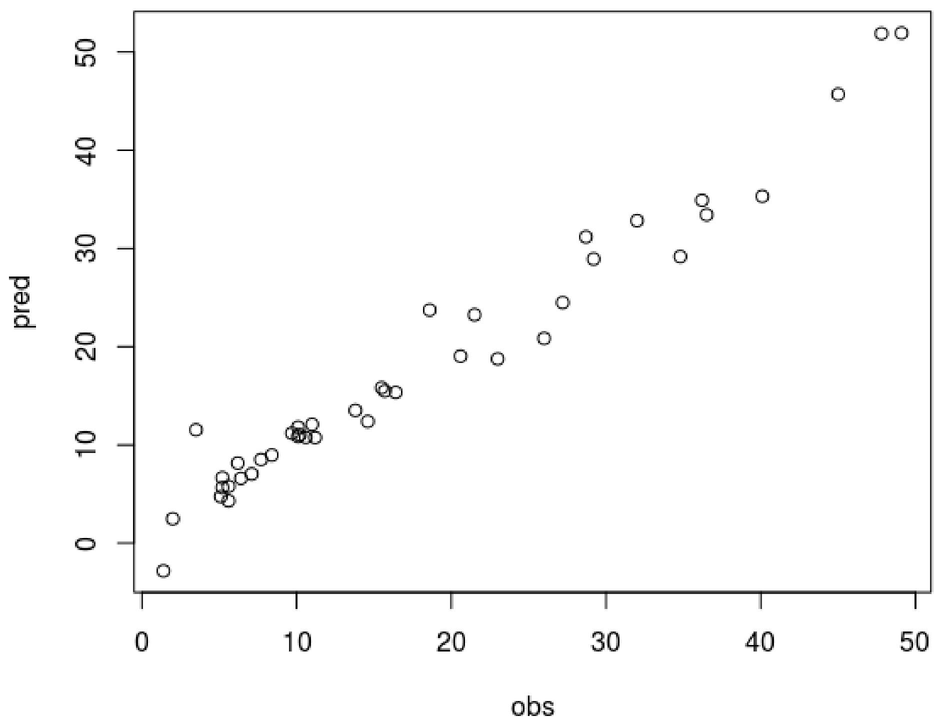

Figure 15: MARS Accuracy Plot: Prediction - Observation Tuned

```
> svmRTuned
Support Vector Machines with Radial Basis Function Kernel

174 samples
100 predictors

Pre-processing: centered (100), scaled (100)
Resampling: Cross-Validated (10 fold, repeated 4 times)
Summary of sample sizes: 158, 156, 157, 156, 157, 157, ...
Resampling results across tuning parameters:

  C        RMSE      Rsquared   MAE
     0.25  9.159371  0.5494555  6.974791
     0.50  7.902765  0.6308165  5.857767
     1.00  6.721085  0.7203609  4.779149
     2.00  5.781818  0.7842770  3.955716
     4.00  5.143081  0.8278796  3.489450
     8.00  4.763295  0.8476536  3.154953
    16.00  4.715718  0.8485002  3.083285
    32.00  4.728965  0.8476388  3.060437
    64.00  4.728111  0.8476738  3.057443
   128.00  4.728111  0.8476738  3.057443
   256.00  4.728111  0.8476738  3.057443
   512.00  4.728111  0.8476738  3.057443
  1024.00  4.728111  0.8476738  3.057443
  2048.00  4.728111  0.8476738  3.057443

Tuning parameter 'sigma' was held constant at a value of 0.194723
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.194723 and C = 16.
```

Figure 16: SVM-Radial Tuned Model Summary

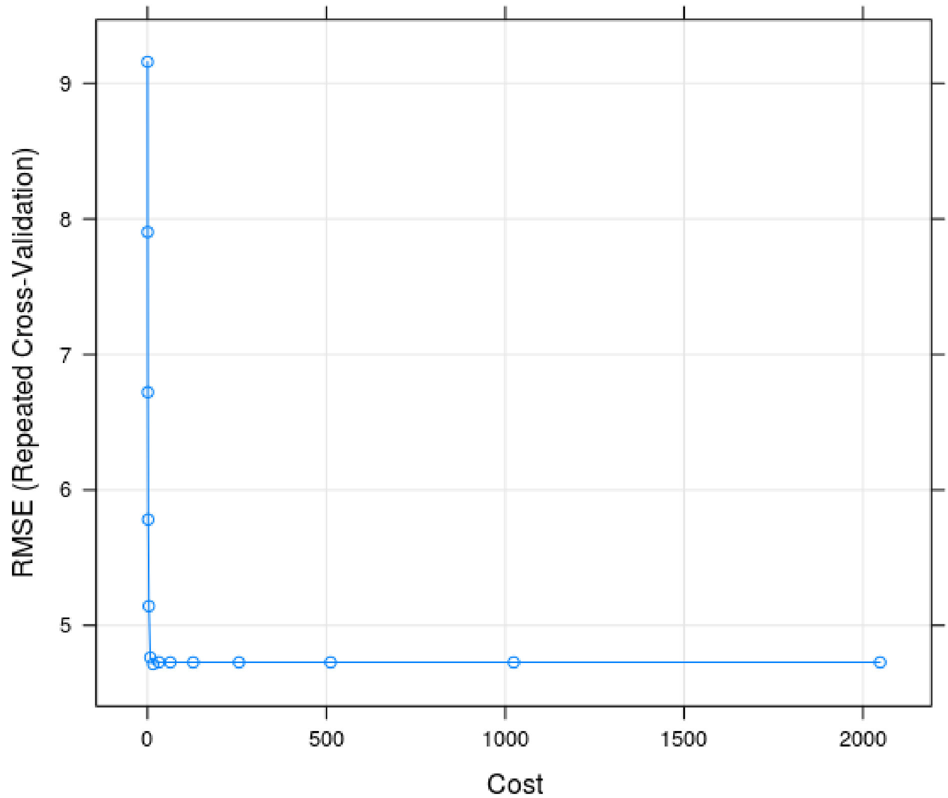

Figure 17: SVM-Radial Tuned Model Plot

```
> defaultSummary(accuracy)
     RMSE  Rsquared       MAE
4.7586057 0.8718353 3.2123679
```

Figure 18: SVM-Radial Accuracy Summary: Prediction - Observation

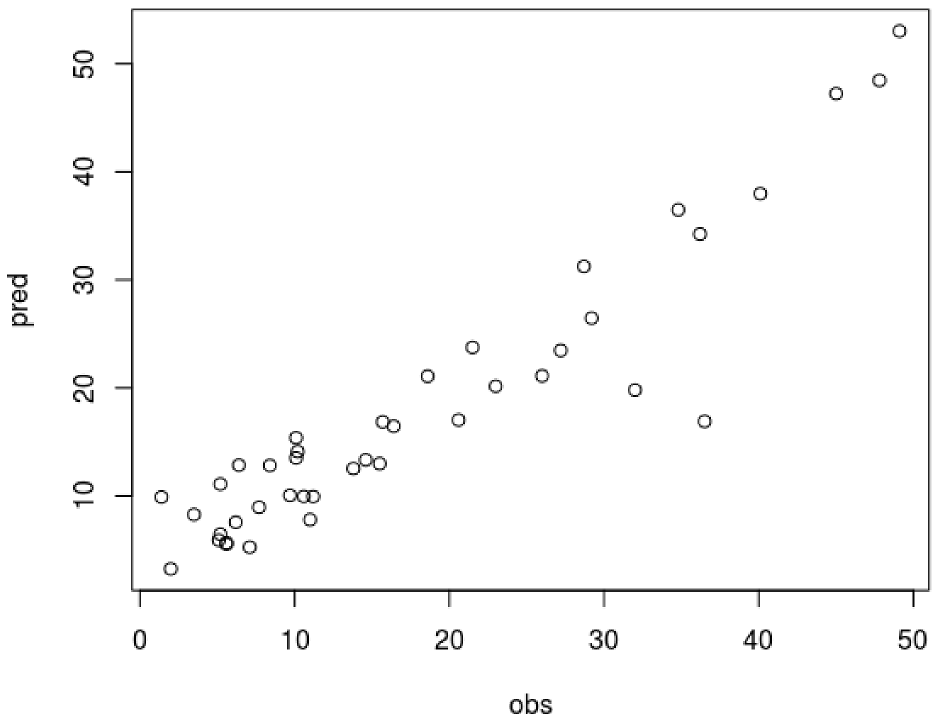

Figure 19: SVM-Radial Accuracy Plot: Prediction - Observation

```
> knnTune
k-Nearest Neighbors

174 samples
100 predictors

Pre-processing: centered (100), scaled (100)
Resampling: Cross-Validated (10 fold, repeated 4 times)
Summary of sample sizes: 156, 155, 158, 156, 158, 156, ...
Resampling results across tuning parameters:

  k   RMSE       Rsquared   MAE
   1   8.315091  0.6033134  5.532095
   2   8.946999  0.5135587  6.500513
   3   9.457958  0.4483264  6.993065
   4   9.574848  0.4345862  7.217432
   5   9.258134  0.4696044  7.150124
   6   9.287884  0.4628565  7.180238
   7   9.267220  0.4636869  7.232700
   8   9.348243  0.4563711  7.310418
   9   9.406116  0.4532838  7.396450
  10   9.466895  0.4524767  7.470513
  11   9.536078  0.4458955  7.521826
  12   9.677400  0.4286948  7.701187
  13   9.718816  0.4245542  7.772968
  14   9.844575  0.4116909  7.855709
  15   9.900428  0.4079200  7.942295
  16  10.015054  0.3956567  8.050392
  17  10.063584  0.3917416  8.158193
  18  10.095777  0.3870240  8.226058
  19  10.136594  0.3850966  8.253951
  20  10.208160  0.3729783  8.316868

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 1.
```

Figure 20: KNN Model Summary

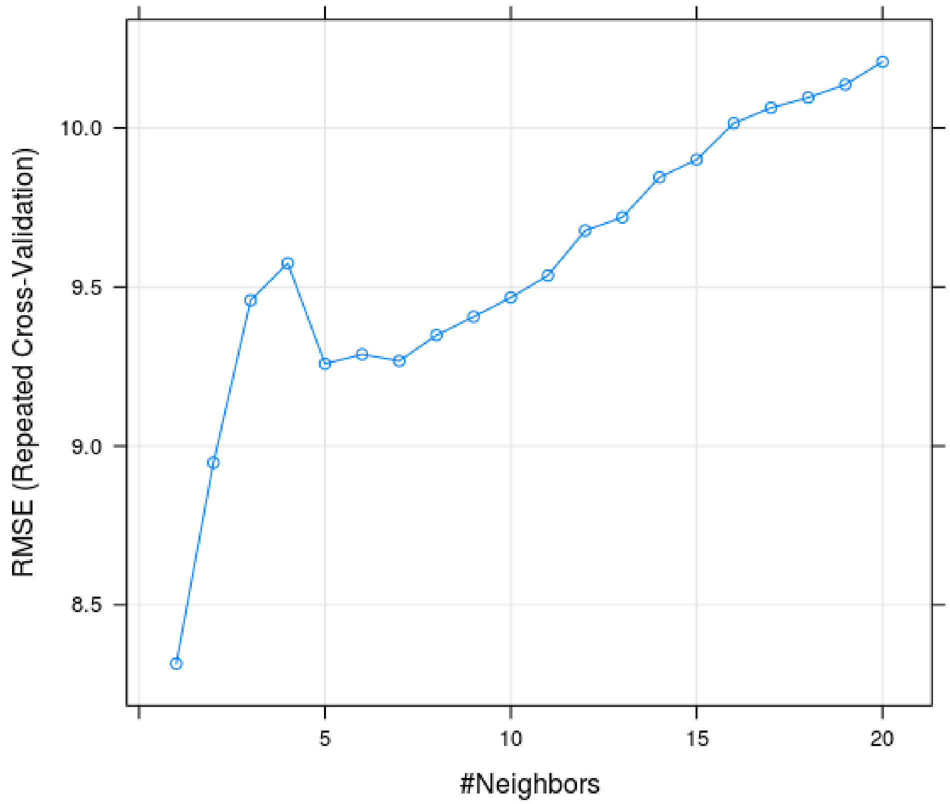

Figure 21: KNN Model Plot

```
> defaultSummary(accuracy)
     RMSE  Rsquared       MAE
8.7544692 0.6142247 5.8268293
```

Figure 22: KNN Accuracy Summary: Prediction - Accuracy

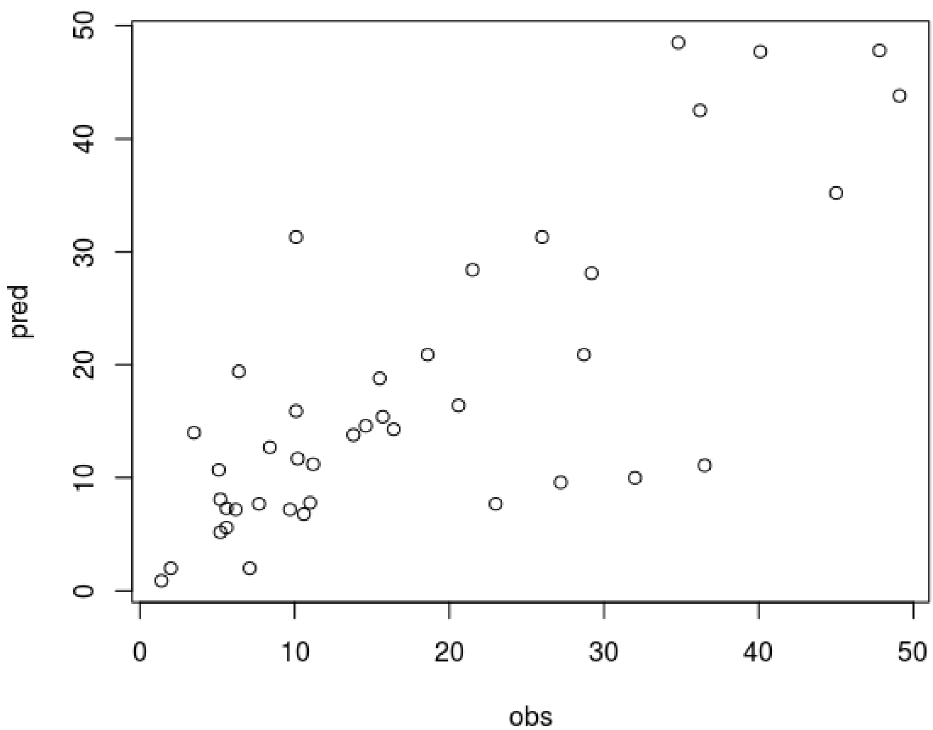

Figure 23: KNN Accuracy Plot: Prediction - Observation

```
> nnetTune2
Model Averaged Neural Network

174 samples
100 predictors

Pre-processing: centered (100), scaled (100), principal component signal extraction (100)
Resampling: Cross-Validated (10 fold, repeated 4 times)
Summary of sample sizes: 156, 158, 155, 158, 156, 157, ...
Resampling results across tuning parameters:

  decay  size  RMSE      Rsquared   MAE
  0.00    1    10.83870  0.2795124   8.670092
  0.00    2    10.73860  0.2915929   8.424960
  0.00    3    10.69589  0.3015033   8.309019
  0.00    4    10.92345  0.2799197   8.470318
  0.00    5    11.17521  0.2800460   8.645128
  0.00    6    10.94161  0.2844724   8.589594
  0.00    7    11.25642  0.2680606   8.765662
  0.00    8    12.68892  0.2272815   9.397043
  0.00    9    14.69912  0.2589522   9.921338
  0.00   10    18.27395  0.2584183  11.496480
  0.01    1    10.91665  0.2665854   8.751205
  0.01    2    10.71787  0.2934550   8.358270
  0.01    3    10.75124  0.2970781   8.375123
  0.01    4    10.87459  0.2855726   8.526499
  0.01    5    10.88645  0.2851492   8.498126
  0.01    6    10.93441  0.2865586   8.494218
  0.01    7    10.94533  0.2912389   8.476587
  0.01    8    11.19072  0.2686477   8.723665
  0.01    9    11.03901  0.2906250   8.610085
  0.01   10    11.37302  0.2744310   8.795411
  0.10    1    10.80948  0.2835030   8.614255
  0.10    2    10.74956  0.2890382   8.390946
  0.10    3    10.75325  0.2941853   8.381084
  0.10    4    10.77852  0.2970194   8.434267
  0.10    5    10.88193  0.2903302   8.459108
  0.10    6    10.85267  0.2877516   8.501294
  0.10    7    10.95242  0.2923130   8.553463
  0.10    8    11.10633  0.2755902   8.600959
  0.10    9    11.14485  0.2921189   8.562459
  0.10   10    11.01462  0.2923798   8.570402

Tuning parameter 'bag' was held constant at a value of FALSE
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 3, decay = 0 and bag = FALSE.
```

Figure 24: Neural Net Model using PCA Model Summary

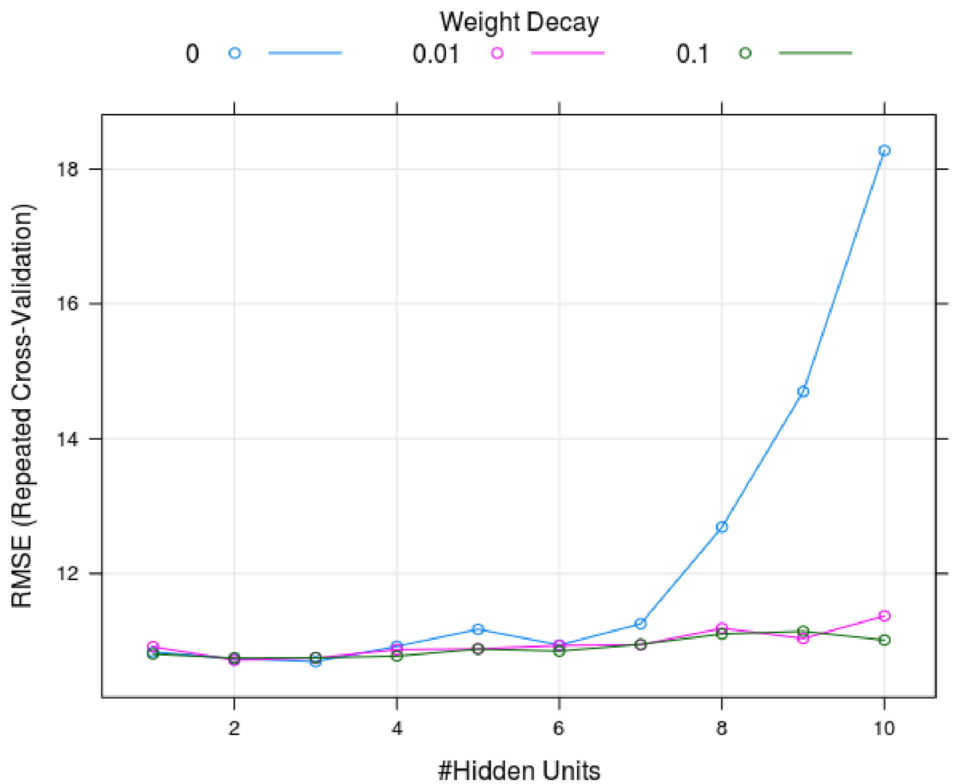

Figure 25: Neural Net Model using PCA Plot

```
> defaultSummary(accuracy)
      RMSE   Rsquared        MAE
11.2388264  0.2874185  9.0059724
```

Figure 26: Neural Net Model using PCA Accuracy Summary: Prediction - Accuracy

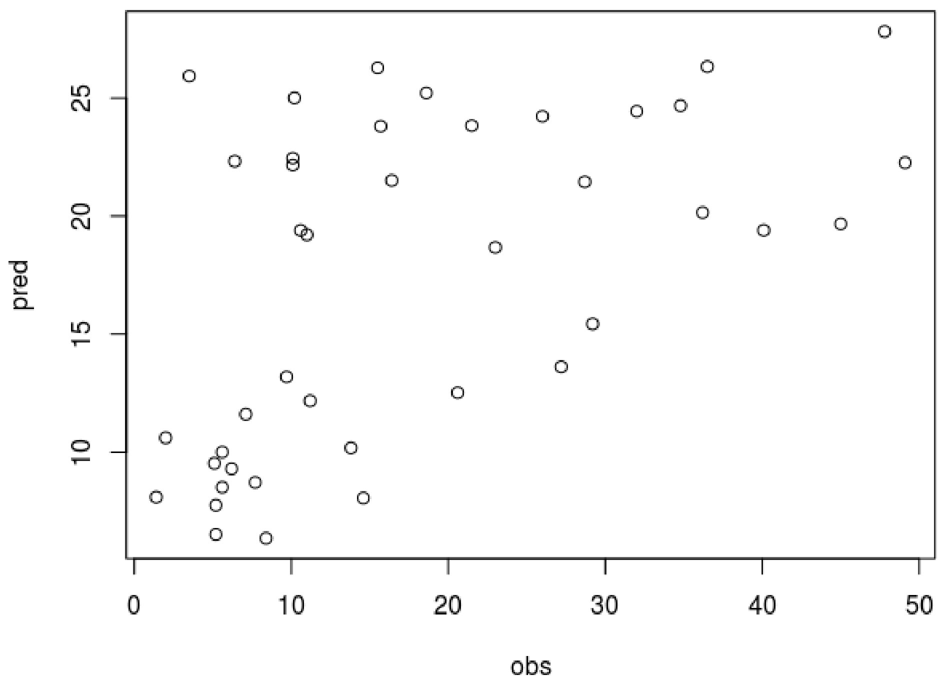

Figure 27: Neural Net Model using PCA Accuracy Plot: Prediction - Observation

```
> nnetTune1
Model Averaged Neural Network

174 samples
100 predictors

Pre-processing: centered (100), scaled (100)
Resampling: Cross-Validated (10 fold, repeated 4 times)
Summary of sample sizes: 156, 158, 155, 158, 156, 157, ...
Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared   MAE
  0.00    1    2.8877532  0.9550868  2.1164489
  0.00    2    1.5191685  0.9849586  1.0409566
  0.00    3    1.2448314  0.9900064  0.8168203
  0.00    4    0.9392078  0.9931484  0.6148081
  0.00    5    1.0653826  0.9914332  0.6603139
  0.00    6    0.9151951  0.9937225  0.5922712
  0.00    7    1.0874712  0.9904128  0.6442169
  0.00    8    1.0035209  0.9928381  0.6072569
  0.00    9    1.2596947  0.9831212  0.7056904
  0.00   10    1.2642047  0.9862034  0.7101381
  0.01    1    1.5982825  0.9849071  1.2402142
  0.01    2    0.8571883  0.9955296  0.6325975
  0.01    3    0.6029053  0.9978428  0.4448570
  0.01    4    0.5278990  0.9983522  0.3897644
  0.01    5    0.5245062  0.9982806  0.3765550
  0.01    6    0.5585063  0.9982176  0.4065915
  0.01    7    0.5079580  0.9984466  0.3749007
  0.01    8    0.5778608  0.9980257  0.4208254
  0.01    9    0.5844420  0.9980020  0.4233049
  0.01   10    0.5782168  0.9979847  0.4122499
  0.10    1    1.9746139  0.9767229  1.6103508
  0.10    2    0.9570141  0.9946615  0.7502700
  0.10    3    0.7379542  0.9968783  0.5930685
  0.10    4    0.7343200  0.9969130  0.5881364
  0.10    5    0.7269771  0.9969529  0.5774530
  0.10    6    0.7223329  0.9969911  0.5714079
  0.10    7    0.7510525  0.9966969  0.5858310
  0.10    8    0.7594181  0.9966299  0.5918315
  0.10    9    0.7748255  0.9963989  0.6003052
  0.10   10    0.8119919  0.9961887  0.6266793

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 = 7, decay = 0.01 and bag = FALSE.
```

Figure 28: Neural Net Model without using PCA Model Summary

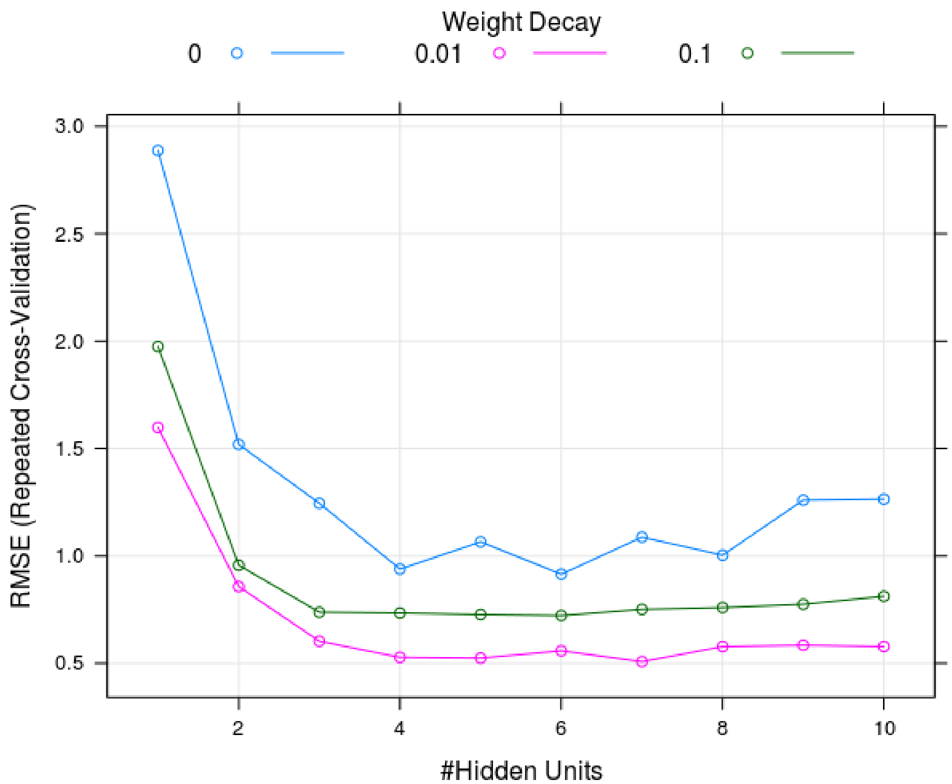

Figure 29: Neural Net without using PCA Plot

```
> defaultSummary(accuracy)
     RMSE  Rsquared       MAE
0.4631235 0.9988229 0.3726964
```

Figure 30: Neural Net Without Using PCA Accuracy Summary: Prediction - Accuracy

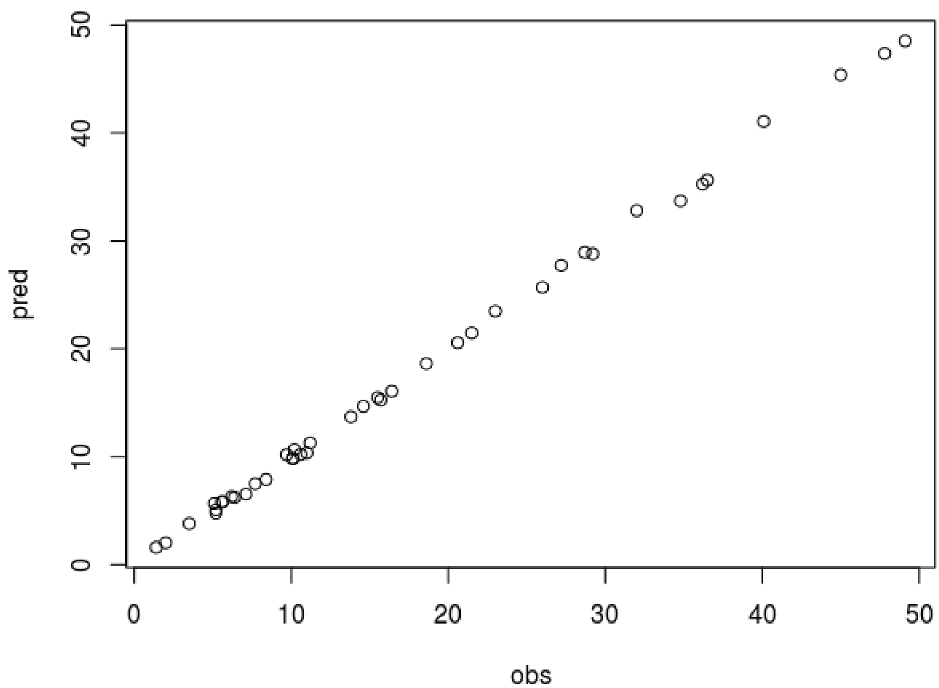

Figure 31: Neural Net Without Using PCA Accuracy Plot: Prediction - Observation

Comparison table for different non-linear model's Performance:

#	Model	RMSE Training	R^2 Training	RMSE Testing	R^2 Testing
1	MARS	2.681071	0.9525823	2.6474251	0.9603677
2	SVM - Radial	4.715718	0.8485002	4.7586057	0.8718353
3	KNN	8.315091	0.6033134	8.754492	0.6142247
4	Neural Net - with PCA	10.69589	0.3015033	11.2388264	0.2874185
5	Neural Net - without PCA	0.5079580	0.99844466	0.4631235	0.9988229

From table, the minimum *RMSE* is given by Neural Net - without PCA which is 0.4631235. So, this is the better model for this data. However, SVM-Radial also gives closely minimum *RMSE* which is 4.7586057. If fast result is wanted, you can go with SVM otherwise, neural network without PCA is good for this data.

Comparing Neural Network with PCA and Neural Network without PCA, Neural Network without PCA seems to work better in this case.

Problem 7.4 :Return to the permeability problem outlined in Exercise 6.2. Train several nonlinear regression models and evaluate the resampling and test set performance.

Solution 7.4:
The Neural Network,MARS, SVM, and KNN models based on data tecator are modeled below. The screenshots of each models are given below which itself explains about their result and analysis. Further, a comparative analysis for each model is at last, made to give brief result of each models

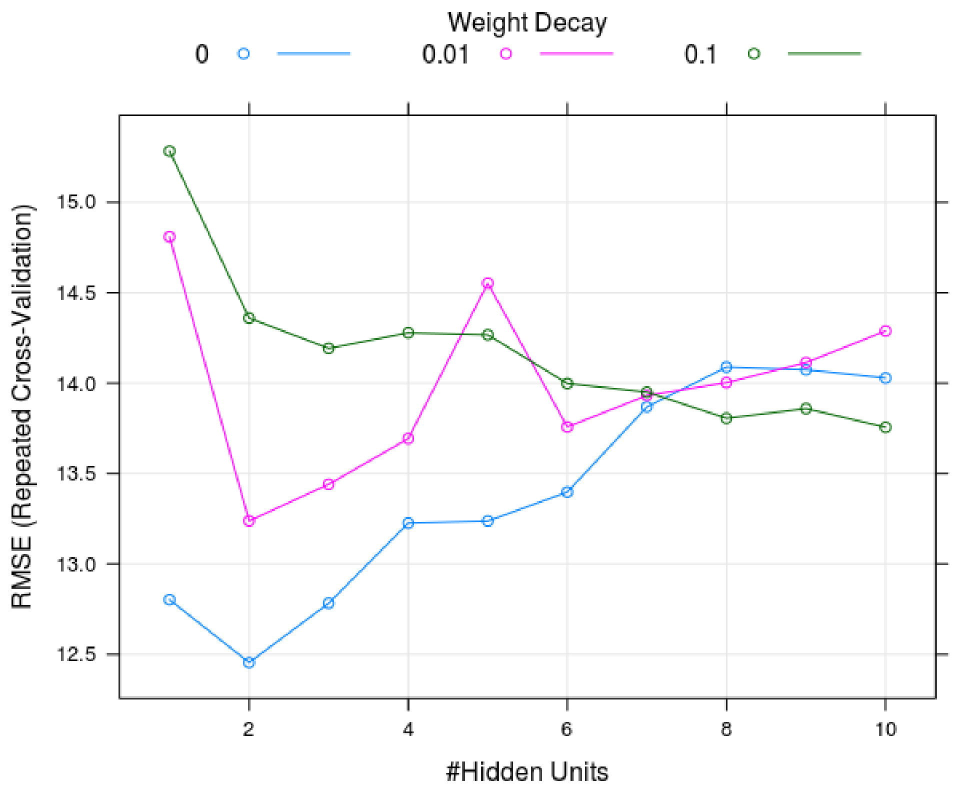

Figure 32: Neural Net Model Plot

```
> nnetTune
Model Averaged Neural Network

125 samples
 57 predictors

Pre-processing: centered (57), scaled (57)
Resampling: Cross-Validated (4 fold, repeated 5 times)
Summary of sample sizes: 93, 93, 94, 95, 94, 93, ...
Resampling results across tuning parameters:

  decay  size  RMSE      Rsquared   MAE
  0.00    1    12.80215  0.3123582   9.494508
  0.00    2    12.45512  0.3680071   9.103036
  0.00    3    12.78280  0.3598360   9.464189
  0.00    4    13.22598  0.3420673   9.748998
  0.00    5    13.23729  0.3254701   9.950389
  0.00    6    13.39743  0.3424400   9.999986
  0.00    7    13.86831  0.3160302  10.495528
  0.00    8    14.08836  0.3111514  10.469850
  0.00    9    14.07374  0.3114438  10.680121
  0.00   10    14.02838  0.3078971  10.747014
  0.01    1    14.80865  0.2774345  10.698270
  0.01    2    13.23783  0.3443188   9.837216
  0.01    3    13.44071  0.3379584  10.006322
  0.01    4    13.69303  0.3264074   9.910754
  0.01    5    14.55184  0.2874800  10.777013
  0.01    6    13.75760  0.3187868  10.167772
  0.01    7    13.93149  0.3110510  10.338550
  0.01    8    14.00273  0.3089000  10.611323
  0.01    9    14.11365  0.3147032  10.589815
  0.01   10    14.28797  0.3007679  10.676174
  0.10    1    15.28263  0.2549981  11.109468
  0.10    2    14.35896  0.2943769  10.462196
  0.10    3    14.19187  0.3037873  10.648478
  0.10    4    14.27777  0.3011878  10.614864
  0.10    5    14.26658  0.2849704  10.659214
  0.10    6    13.99785  0.3193291  10.450871
  0.01    7    13.93149  0.3110510  10.338550
  0.01    8    14.00273  0.3089000  10.611323
  0.01    9    14.11365  0.3147032  10.589815
  0.01   10    14.28797  0.3007679  10.676174
  0.10    1    15.28263  0.2549981  11.109468
  0.10    2    14.35896  0.2943769  10.462196
  0.10    3    14.19187  0.3037873  10.648478
  0.10    4    14.27777  0.3011878  10.614864
  0.10    5    14.26658  0.2849704  10.659214
  0.10    6    13.99785  0.3193291  10.450871
  0.10    7    13.94987  0.3115963  10.506039
  0.10    8    13.80544  0.3233590  10.294200
  0.10    9    13.85862  0.3218990  10.368635
  0.10   10    13.75579  0.3257288  10.252556

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 = 2, decay = 0 and bag = FALSE.
```

Figure 33: Neural Net Model Summary

```
> defaultSummary(accuracy)
     RMSE  Rsquared       MAE
9.4495428 0.7041009 6.5397654
>
```

Figure 34: Neural Net Accuracy Summary: Prediction - Observation Tuned

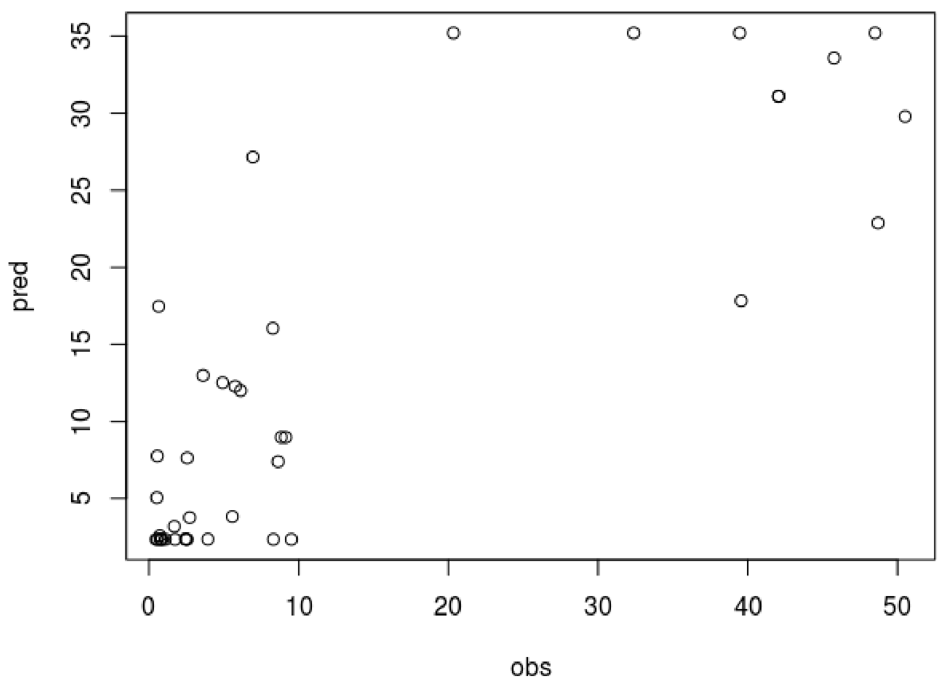

Figure 35: Neural Net Accuracy Plot: Prediction - Observation Tuned

```
> summary(marsFit)
Call: earth(x=trainFingerprints, y=trainPermeability)

             coefficients
(Intercept)     16.243227
X6               8.929049
X15            -24.031343
X87            -25.377771
X97             35.679449
X111            -9.144239
X129            34.143015
X141             6.706075
X143           -14.154159
X258           -14.441528
X338            13.564079
X366           -16.397336
X514           -37.358550
X679           -17.225601

Selected 14 of 55 terms, and 13 of 388 predictors
Termination condition: GRSq -10 at 55 terms
Importance: X129, X6, X15, X141, X698-unused, X97, X143, X258, X366, X514, X679, ...
Number of terms at each degree of interaction: 1 13 (additive model)
GCV 107.4728    RSS 8257.353    GRSq 0.5341149    RSq 0.7090036
```

Figure 36: MARS using internal GCM for model selection summary

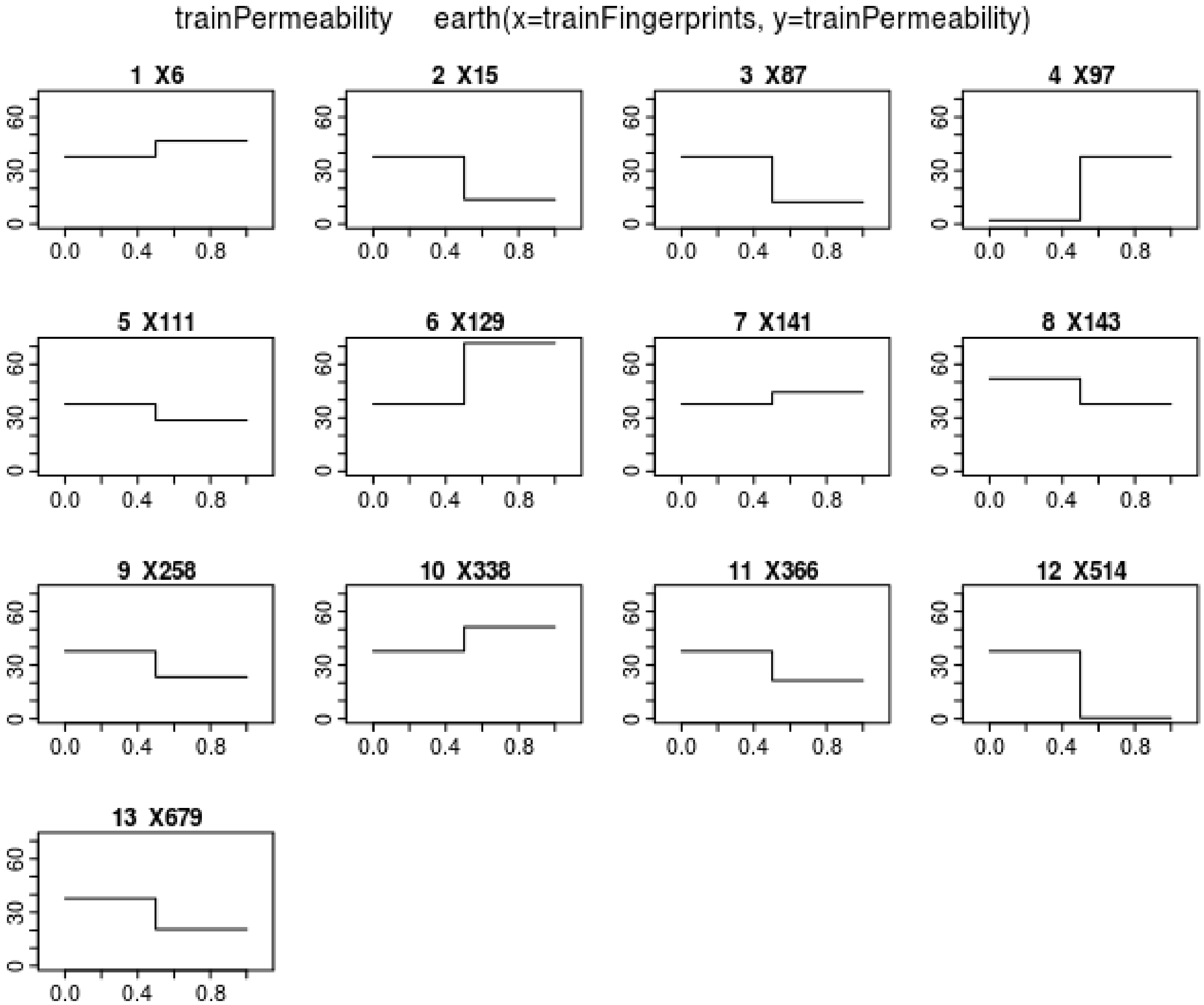

Figure 37: MARS using internal GCM for model selection Plot

```
Multivariate Adaptive Regression Spline

125 samples
388 predictors

No pre-processing
Resampling: Cross-Validated (4 fold, repeated 5 times)
Summary of sample sizes: 93, 93, 94, 95, 95, 94, ...
Resampling results across tuning parameters:

  degree  nprune  RMSE      Rsquared   MAE
  1        2      13.99049  0.1971027  10.432070
  1        3      12.94962  0.3060700   9.024760
  1        4      12.54846  0.3443737   8.523240
  1        5      12.17277  0.3845627   8.217711
  1        6      12.16351  0.3934954   8.308285
  1        7      12.07990  0.4028986   8.285071
  1        8      12.51790  0.3793273   8.587779
  1        9      12.95551  0.3511337   8.977572
  1       10      13.48245  0.3300207   9.348351
  1       11      13.67419  0.3134398   9.501072
  1       12      14.05312  0.2962780   9.787686
  1       13      14.11394  0.2958046   9.789860
  2        2      14.10804  0.1870575  10.486852
  2        3      13.60629  0.2779445   9.554733
  2        4      13.80492  0.2945012   9.427882
  2        5      13.58390  0.3286934   9.183770
  2        6      14.02996  0.3039323   9.301210
  2        7      13.76756  0.3285699   8.935403
  2        8      14.58097  0.2944488   9.420357
  2        9      14.70838  0.2949837   9.443014
  2       10      14.89755  0.2881833   9.567551
  2       11      15.24041  0.2860163   9.796049
  2       12      15.94824  0.2751861  10.238892
  2       13      16.08189  0.2696753  10.324772

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.
```

Figure 38: MARS Tuned Model Summary Using Train()

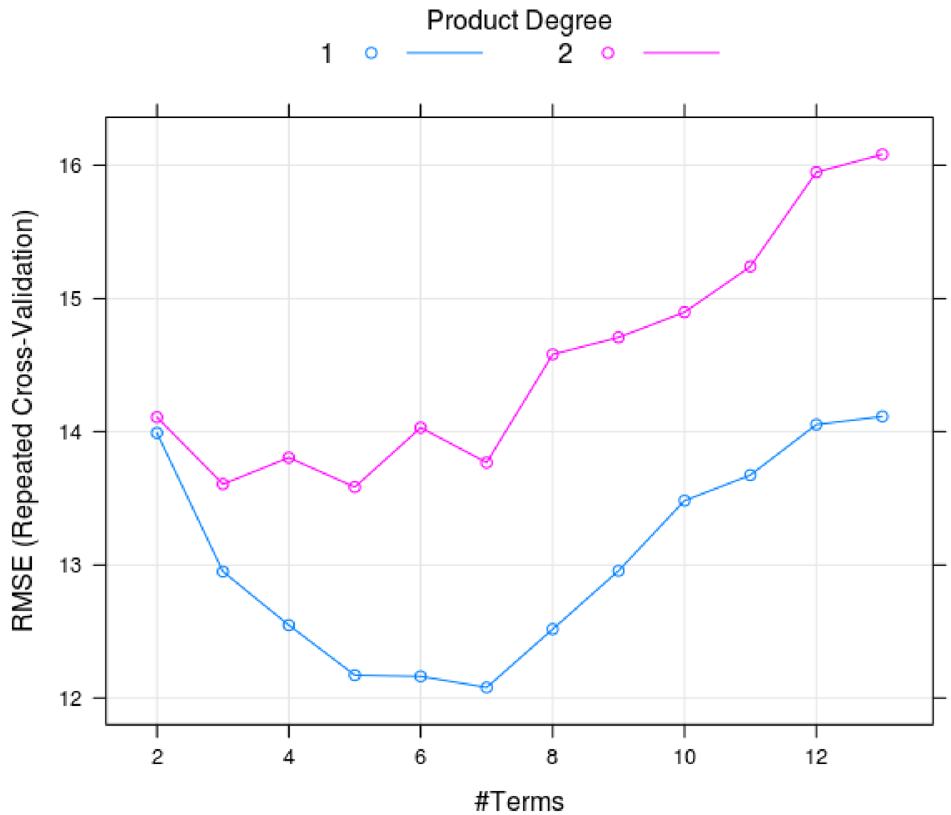

Figure 39: MARS Tuned Model Plot Using Train()

```
> defaultSummary(accuracy)
      RMSE   Rsquared        MAE
12.0544578  0.5031097  8.0067312
```

Figure 40: MARS Accuracy Summary: Prediction - Observation Tuned Using Train()

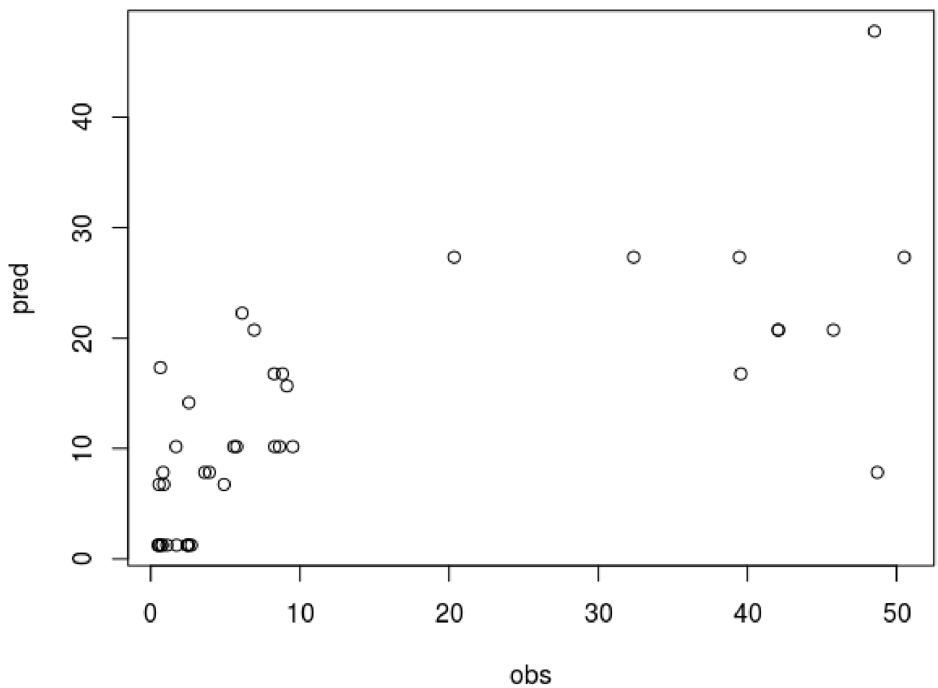

Figure 41: MARS Accuracy Plot: Prediction - Observation Tuned

```
> permeabilitysvmRTuned
Support Vector Machines with Radial Basis Function Kernel

125 samples
388 predictors

Pre-processing: centered (388), scaled (388)
Resampling: Cross-Validated (4 fold, repeated 5 times)
Summary of sample sizes: 93, 93, 96, 93, 94, 94, ...
Resampling results across tuning parameters:

  C        RMSE      Rsquared   MAE
     0.25  13.93550  0.3274944  8.971172
     0.50  12.95863  0.3496293  8.558090
     1.00  12.36019  0.3761192  8.339565
     2.00  11.94639  0.4003177  8.323477
     4.00  11.40963  0.4522293  8.058279
     8.00  11.17340  0.4694920  7.973273
    16.00  10.98640  0.4797654  7.910892
    32.00  11.02139  0.4775127  7.990165
    64.00  11.02966  0.4766114  8.005418
   128.00  11.02969  0.4766075  8.005459
   256.00  11.02966  0.4766092  8.005453
   512.00  11.02966  0.4766090  8.005410
  1024.00  11.02961  0.4766135  8.005425
  2048.00  11.02960  0.4766144  8.005389

Tuning parameter 'sigma' was held constant at a value of 0.001883989
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.001883989 and C = 16.
```

Figure 42: SVM-Radial Tuned Model Summary

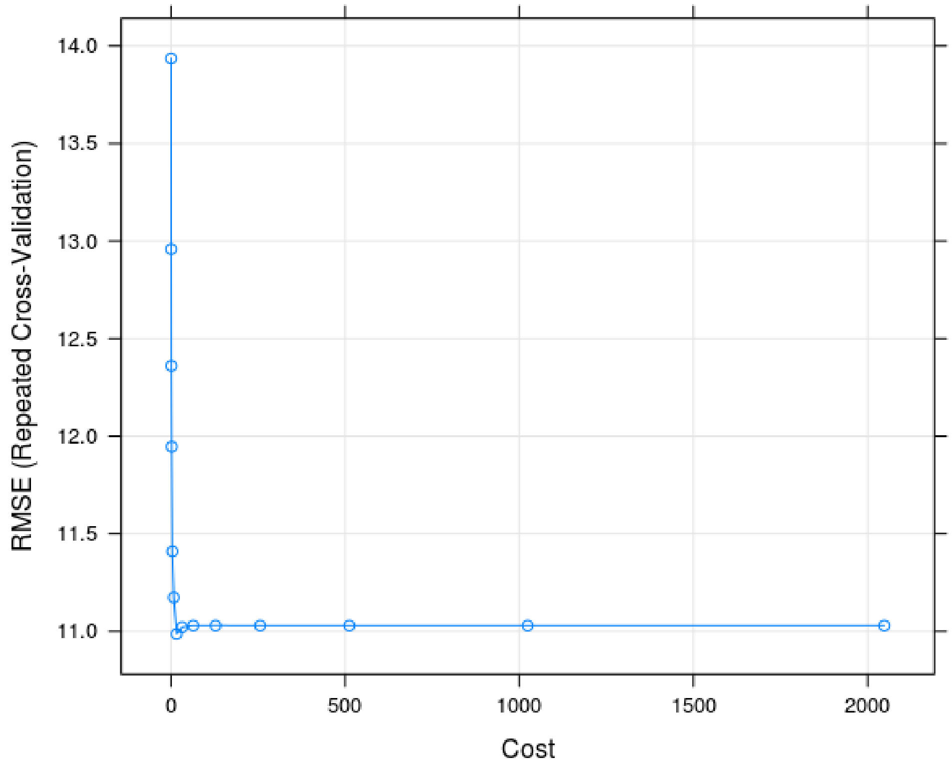

Figure 43: SVM-Radial Tuned Model Plot

```
> defaultSummary(accuracy)
      RMSE   Rsquared        MAE
10.0145041  0.6999508  7.2781667
```

Figure 44: SVM-Radial Accuracy Summary: Prediction - Observation

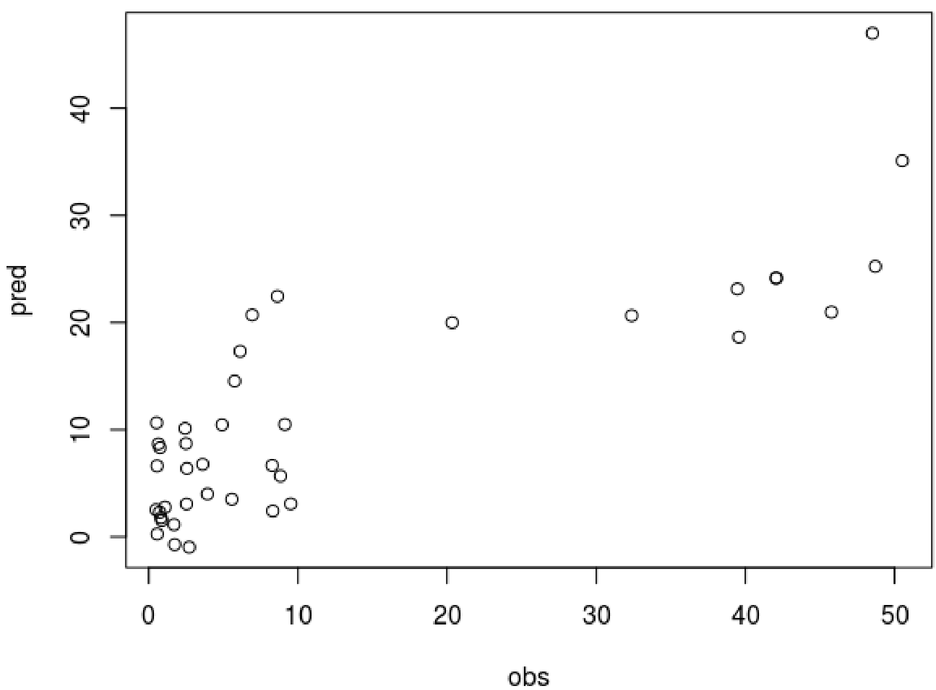

Figure 45: SVM-Radial Accuracy Plot: Prediction - Observation

```
> permeabilityknnTuned
k-Nearest Neighbors

125 samples
377 predictors

Pre-processing: centered (377), scaled (377)
Resampling: Cross-Validated (4 fold, repeated 5 times)
Summary of sample sizes: 93, 93, 96, 93, 94, 94, ...
Resampling results across tuning parameters:

  k   RMSE      Rsquared   MAE
   1  13.63794  0.3269161   8.816513
   2  13.63864  0.2938634   9.021412
   3  13.02715  0.3033846   8.828082
   4  12.85670  0.3149996   8.691254
   5  12.82840  0.3068922   8.632893
   6  12.89567  0.2918556   8.691110
   7  12.88980  0.2924890   8.761552
   8  12.90222  0.2878102   8.876377
   9  12.89917  0.2873193   8.959847
  10  12.96870  0.2767568   9.108224
  11  13.08033  0.2641488   9.264221
  12  13.19245  0.2509004   9.477296
  13  13.29329  0.2373666   9.654478
  14  13.40996  0.2261885   9.824832
  15  13.52477  0.2130598   9.973251
  16  13.57602  0.2099075  10.069857
  17  13.64722  0.2000482  10.186227
  18  13.72674  0.1897893  10.308082
  19  13.74555  0.1872570  10.338827
  20  13.79299  0.1823410  10.413042

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 5.
```

Figure 46: KNN Model Summary

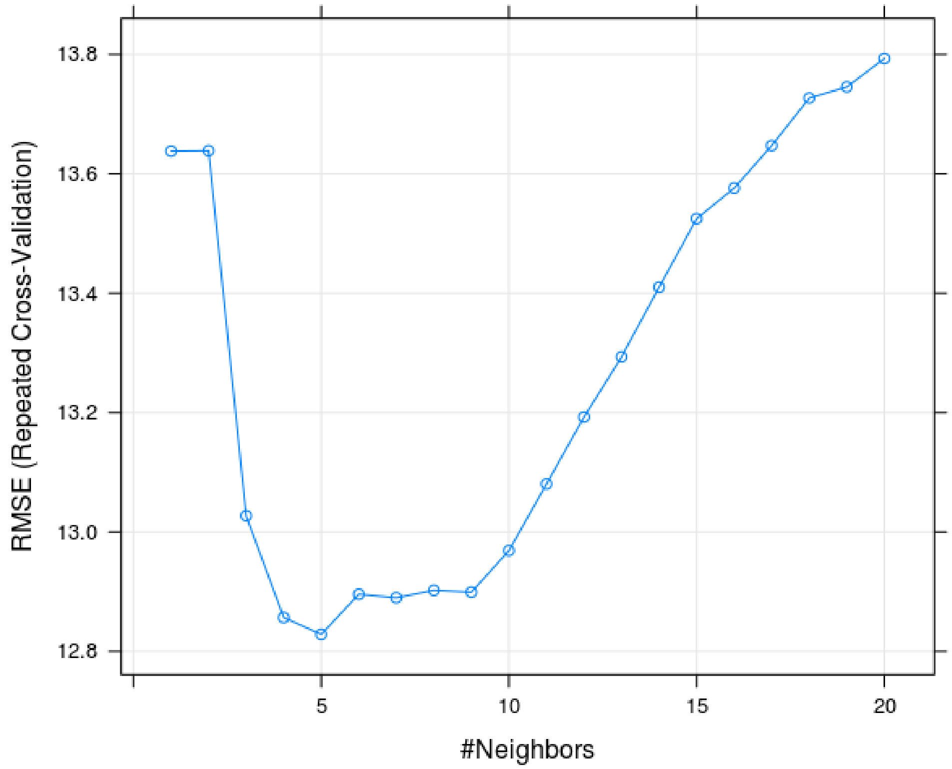

Figure 47: KNN Model Plot

```
> defaultSummary(accuracy)
     RMSE  Rsquared       MAE
9.8862562 0.6831458 7.7392250
```

Figure 48: KNN Accuracy Summary: Prediction - Accuracy

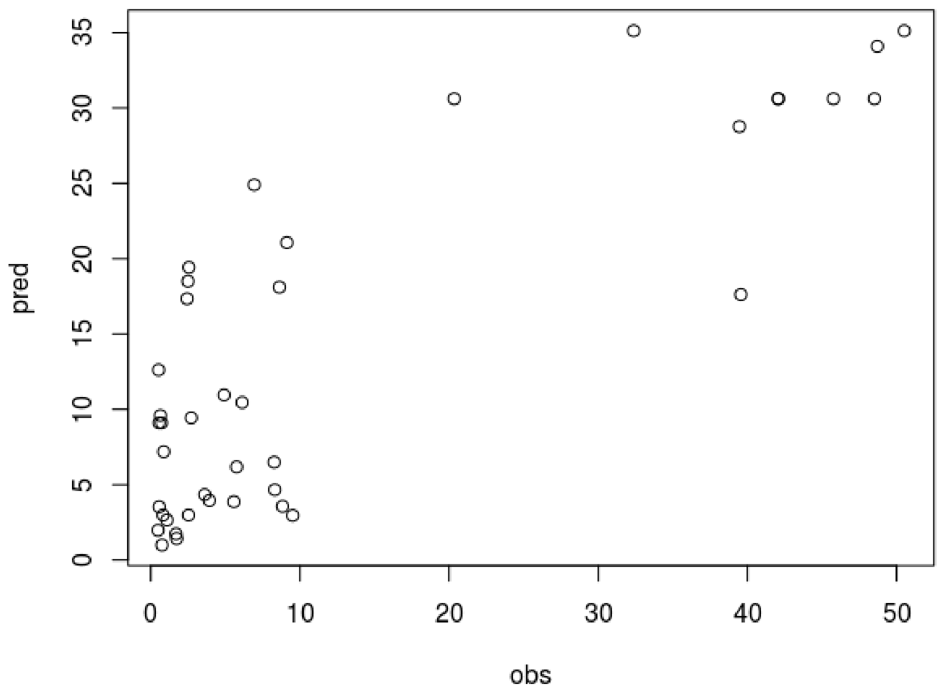

Figure 49: KNN Accuracy Plot: Prediction - Observation

Comparison table for different Linear and Non-Linear Model's Performance

1	Model	RMSE Training	R^2 Training	RMSE Testing	R^2 Testing
2	PLS	12.61378	0.3761673	10.8629421	0.5957486
3	Ridge	13.277328	0.3812073	10.4881743	0.6284279
4	Elastic Net	12.12502	0.4011406	10.0757239	0.6605789
5	Neural Net	12.45512	0.3680071	9.4495428	0.7041009
6	MARS	12.07990	0.4028986	12.0544578	0.5031097
7	SVM Radial	10.98649	0.4797654	10.0145041	0.6999508
8	KNN	12.82840	0.3068922	9.8862562	0.6831458

Answer to 7.4(a):
From these nonlinear models, MARS model performs best with R^2 value of 0.5031097. This is best than PLS model which has R^2 value of 0.5957486. Hence, the results indicate that underlying relationship between the predictor and response is likely best described by a non-linear structure .

Answer to 7.4(b):
From table, the minimum $RMSE$ is given by Neural Net model which is 9.4495428. So, this is the better model for this data. However, KNN also gives closely minimum $RMSE$ which is 9.88262562. If fast result is wanted, you can go with KNN otherwise, neural network is good for this data.

Answer to 7.4(c):
Comparing result of $RMSE$ and R^2 so far, KNN outperforms the best. And we would recommend KNN model for the model tuned so far.

```
########################
# question 7.1
########################

library(caret)
library(kernlab)
library(lattice)
library(ggplot2)

x = runif(100,min=2,max=10)
y = sin(x)+rnorm(length(x))*0.25
sinData = data.frame(x=x,y=y)

# create a data Grid
dataGrid = data.frame(x=seq(2,10,length=100))

# this is done to divide the graph in 4 columns
par(mfrow = c(2,2))

svmParam1 = expand.grid(eps = c(0.01,0.05,0.1,0.5), costs =  2 c(-2,0,2,8))
for ( i in 1: nrow(svmParam1))-
  set.seed(121)
  rbfSVM = ksvm(x=x,y=y, data=sinData,
                 kernel=”rbfdot”,kpar=”automatic”,
                 C= svmParam1$costs[i],epsilon = svmParam1$eps[i])

  tmp = data.frame(x=dataGrid$x,y =predict(rbfSVM,newdata= dataGrid),
                  eps=paste(”epsilon”,format(svmParam1$eps)[i]),
                  costs=paste(”costs”,format(svmParam1$costs)[i]))

  svmPred1 = if(i==1) tmp else rbind(tmp,svmPred1)

  modelPrediction = predict(rbfSVM, newdata = dataGrid)
  plot(x,y)
  points(x = dataGrid$x, y = modelPrediction[,1], type = ”l”, col = ”blue”)


svmPred1$costs = factor(svmPred1$costs, levels=rev(levels(svmPred1$costs)))


svmParam2 = expand.grid(eps = c(0.01,0.05,0.1,0.5),
                         costs =  2 c(-2,0,2,8),
                         sigma=as.vector(sigest(y x,data=sinData,frac=.75)))

for ( i in 1: nrow(svmParam2))-
  set.seed(121)
  rbfSVM = ksvm(x=x,y=y, data=sinData,
                 kernel=”rbfdot”,
                 kpar=list(sigma=svmParam2$sigma[i]),
                 C= svmParam2$costs[i],
                 epsilon = svmParam2$eps[i]
                 )

  tmp = data.frame(x=dataGrid$x,
                  y =predict(rbfSVM,newdata= dataGrid),
                  eps=paste(”epsilon”,format(svmParam2$eps)[i]),
                  costs=paste(”costs”,format(svmParam2$costs)[i]),
                  sigma=paste(”sigma”,format(svmParam2$sigma,digits=2)[i])
  )
  svmPred2 = if(i==1) tmp else rbind(tmp,svmPred2)

  modelPrediction = predict(rbfSVM, newdata = dataGrid)
  plot(x,y)
  points(x = dataGrid$x, y = modelPrediction[,1], type = ”l”, col = ”blue”)
```

```
svmPred2$costs = factor(svmPred2$costs, levels=rev(levels(svmPred2$costs)))
svmPred2$sigma = factor(svmPred2$sigma, levels=rev(levels(svmPred2$sigma)))

###################################
###################################




##################################
#Question 7.3
##################################
library(mlbench)
library(caret)
library(earth)
library(doParallel)
library(nnet)

data(tecator)

colName = –
for (i in 1:100)–
  colName[i]¡- paste("X",i)

colnames(absorp)¡-colName


# splitting data into 80% and 20% based on Fat Response
set.seed(12345)

trainingRows =  createDataPartition(endpoints[,2], p = .80, list= FALSE)

trainAbsorption ¡- absorp[ trainingRows, ]
testAbsorption ¡- absorp[-trainingRows, ]
trainFat ¡- endpoints[trainingRows, 2]
testFat ¡- endpoints[-trainingRows, 2]

ctrl ¡- trainControl(method = "repeatedcv", repeats=4)

# # For neuralnetwork, find the correlation and delete the correlated data
tooHigh ¡- findCorrelation(cor(trainAbsorption), cutoff = .80)

#  the tooHigh gives 99 correlated datas
trainXnnet1 = trainAbsorption[,-tooHigh]
testXnnet1 = testAbsorption[,-tooHigh]

set.seed(12344)

library(nnet)
library(caret)

# without using PCA
# to train in parallel to 5 processor
cl ¡- makePSOCKcluster(5)
registerDoParallel(cl)

nnetGrid1 ¡- expand.grid(.decay = c(0, 0.01, .1),
                        .size = c(1:10),
                        ## The next option is to use bagging (see the
                        ## next chapter) instead of different random
                        ## seeds.
                        .bag = FALSE)
```

```
nnetTune1 ¡- train(trainAbsorption, trainFat,
                  method = ”avNNet”,
                  trControl = ctrl,
                  preProc = c(”center”, ”scale”),
                  linout = TRUE,
                  trace = FALSE,
                  MaxNWts = 10 * (ncol(trainAbsorption) + 1) + 10 + 1,
                  maxit = 500,
                  tuneGrid = nnetGrid1)

prediction¡-predict(nnetTune1,testAbsorption)
accuracy¡-data.frame(obs=testFat,pred=prediction)
defaultSummary(accuracy)
plot(accuracy)
## When you are done:
stopCluster(cl)

# using PCA
# to train in parallel to 5 processor
cl ¡- makePSOCKcluster(5)
registerDoParallel(cl)

nnetGrid1 ¡- expand.grid(.decay = c(0, 0.01, .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(trainAbsorption, trainFat,
                   method = ”avNNet”,
                   trControl = ctrl,
                   preProc = c(”center”, ”scale”,”pca”),
                   linout = TRUE,
                   trace = FALSE,
                   MaxNWts = 10 * (ncol(trainAbsorption) + 1) + 10 + 1,
                   maxit = 500,
                   tuneGrid = nnetGrid1)

prediction¡-predict(nnetTune2,testAbsorption)
accuracy¡-data.frame(obs=testFat,pred=prediction[-41])
defaultSummary(accuracy)
plot(accuracy)
## When you are done:
stopCluster(cl)



# # For MARS, using resampling method to tune the model  Selection Using GCV
set.seed(12345)
marsFit ¡- earth(trainAbsorption,trainFat)
summary(marsFit)
#
set.seed(12345)
marsGrid ¡- expand.grid(.degree = 1:2, .nprune = 2:18)
marsTuned ¡- train(trainAbsorption, trainFat, method=”earth”,
                   tuneGrid = marsGrid,
                   trControl = ctrl)

prediction¡-predict(marsTuned,testAbsorption)
accuracy¡-data.frame(obs=testFat,pred=prediction[,1])
defaultSummary(accuracy)
plot(accuracy)
#
```

```
# # For SVM, using radial function is automatic and if the data are linear in regression should use
# linear svm, otherwise radial SVM is good
set.seed(12345)
svmRTuned ¡- train(trainAbsorption, trainFat, method="svmRadial",
                   tuneLength = 14,
                   preProc = c("center", "scale"),
                   trControl = ctrl)
#
prediction¡-predict(svmRTuned,testAbsorption)
accuracy¡-data.frame(obs=testFat,pred=prediction)
defaultSummary(accuracy)
plot(accuracy)


# # For KNN, remove the near-zero-variance predictors
# # And, do the centering and scaling
knnDescr ¡- trainAbsorption[ ,-nearZeroVar(trainAbsorption)]
set.seed(12345)
knnTune ¡- train(trainAbsorption,trainFat,
                 method="knn",
                 preProc = c("center","scale"),
                 tuneGrid = data.frame(k=1:20),
                 trControl = ctrl)
#
prediction¡-predict(knnTune,testAbsorption)
accuracy¡-data.frame(obs=testFat,pred=prediction)
defaultSummary(accuracy)
plot(accuracy)


###################################
###################################

#########################
# question 7.4
#########################
library(AppliedPredictiveModeling)
library(mlbench)
library(caret)
library(earth)
library(MASS)
library(elasticnet)
library(lars)
library(pls)
library(doParallel)
library(nnet)

data(permeability)


cat("After Non-Zero Variance, number of predictors in fingerprints is 388: "n")
NZVfingerprints ¡- nearZeroVar(fingerprints)
noNZVfingerprints ¡- fingerprints[,-NZVfingerprints]
print(str(noNZVfingerprints))
cat(""n"n")

# stratified random sample splitting with 75% training and 25% testing

set.seed(12345)
trainingRows =  createDataPartition(permeability, p = .75, list= FALSE)
trainFingerprints ¡- noNZVfingerprints[trainingRows,]
trainPermeability ¡- permeability[trainingRows,]

testFingerprints ¡- noNZVfingerprints[-trainingRows,]
```

```
testPermeability ¡- permeability[-trainingRows,]

set.seed(12345)

ctrl ¡- trainControl(method = ”repeatedcv”, repeats=5, number = 4)


# # For neuralnetwork, find the correlation and delete the correlated data
tooHigh ¡- findCorrelation(cor(trainFingerprints), cutoff = .75)
#
# #  the tooHigh gives 99 correlated datas
trainXnnet = trainFingerprints[,-tooHigh]
testXnnet = testFingerprints[,-tooHigh]
#
# set.seed(12344)

nnetGrid ¡- expand.grid(.decay = c(0, 0.01, .1),
                        .size = c(1:10),
                        ## The next option is to use bagging (see the
                        ## next chapter) instead of different random
                        ## seeds.
                        .bag = FALSE)


nnetTune ¡- train(trainXnnet, trainFat,
                  method = ”avNNet”,
                  tuneGrid = nnetGrid,
                  trControl = ctrl,
                  ## Automatically standardize data prior to modeling
                  ## and prediction
                  preProc = c(”center”, ”scale”),
                  linout = TRUE,
                  trace = FALSE,
                  MaxNWts = 10 * (ncol(trainXnnet) + 1) + 10 + 1,
                  maxit = 500)

prediction¡-predict(nnetTune,testXnnet)
accuracy¡-data.frame(obs=testPermeability,pred=prediction)
defaultSummary(accuracy)
plot(accuracy)


# # For MARS, using resampling method to tune the model  Selection Using GCV
set.seed(12345)
marsFit ¡- earth(trainFingerprints,trainPermeability)
summary(marsFit)

set.seed(12345)
permeabilitymarsGrid ¡- expand.grid(degree = 1:2,nprune = 2:13)
permeabilitymarsTuned ¡- train(trainFingerprints, trainPermeability,
                   method=”earth”,
                   tuneGrid = permeabilitymarsGrid,
                   trControl = ctrl)
#

prediction¡-predict(permeabilitymarsTuned,testFingerprints)
accuracy¡-data.frame(obs=testPermeability,pred=prediction[,1])
defaultSummary(accuracy)
plot(accuracy)

#

# # For SVM, using radial function is automatic and if the data are linear in regression should use
# linear svm, otherwise radial SVM is good
set.seed(12345)
```

```
permeabilitysvmRTuned ¡- train(trainFingerprints, trainPermeability,
                               method="svmRadial",
                               tuneLength = 14,
                               preProc = c("center", "scale"),
                               trControl = ctrl)
#
prediction¡-predict(permeabilitysvmRTuned,testFingerprints)
accuracy¡-data.frame(obs=testPermeability,pred=prediction)
defaultSummary(accuracy)
plot(accuracy)


# # For KNN, remove the near-zero-variance predictors
# # And, do the centering and scaling
permeabilityknnDescr ¡- trainFingerprints[ ,-nearZeroVar(trainFingerprints)]
set.seed(12345)
permeabilityknnTuned ¡- train(permeabilityknnDescr,trainPermeability,
                 method="knn",
                 preProc = c("center","scale"),
                 tuneGrid = data.frame(k=1:20),
                 trControl = ctrl)

prediction¡-predict(permeabilityknnTuned,testFingerprints)
accuracy¡-data.frame(obs=testPermeability,pred=prediction)
defaultSummary(accuracy)
plot(accuracy)

#
# #########################
# #########################
#
```

References:

1. Applied Predictive Modeling : @authors Max Kuhn. Kjell Johnson
2. https://archive.ics.uci.edu/ml/index.php