Chapter 7

Chapter 7. Nonlinear Regression Models

7.1 Neural Networks

The outcome is modeled by an intermediary set of unobserved variables (hidden variables or hidden units). These hidden units are linear combinations of the original predictors, but, unlike PLS models, they are not estimated in a hierarchical fashion

Once the number of hidden units is defined, each unit must be related to the outcome. Another linear combination connects the hiddden units to the outcome

Early stopping: the iterative algorithms for solving for the regression equations can be permaturely halted

Weight decay: a penalization method to regularize model similar to ridge regression

Parameters: number of hidden layers, lambda (weight decay)

Predictors should be centered and scaled prior to modeling

7.2 Multivariate Adaptive Regression Splines

Like neural networks and partial least squares, MARS uses surrogate features instead of the original predictors

MARS creates two contrasted versions of a predictor to enter the model. Also, the surrogate features in MARS are usually a function of only one or two predictors at a time. The nature of the MARS features breaks the predictor into two groups and models linear relationships between the predictor and the outcome in each group.

Piecewise linear model: each new feature models an isolated portion of the original data

Hinge function

Two tuning parameters associated with the MARS model: the degree of the features that are added to the model and the number of retained terms

Advantages of MARS: First, the model automatically conducts feature selection, the model equation is independent of predictor variables that are not involved with any of the final model features. Second, interpretability. Each hinge feature is responsible for modeling a specific region in the predictor space using a piecewise linear model. When the MARS model is addictive, the contribution of each predictor can be isolated without the need to consider the others. Third, data transformation and the filtering of predictors are not needed.

The additive nature of the model allows each predictor to be viewed in isolation: changing the values of the other predictor variables will not alter the shape of the profile, only the location on the y-axis where the profile starts

7.3 Support Vector Machines

e-insensitive regression (minimize the effect of outliers on the regression equations.)

Given a threshold set by the user (denoted as e),data points with residuals within the threshold do not contribute to the regression fit while data points with an absolute difference greater than the threshold contribute a linear-scale amount

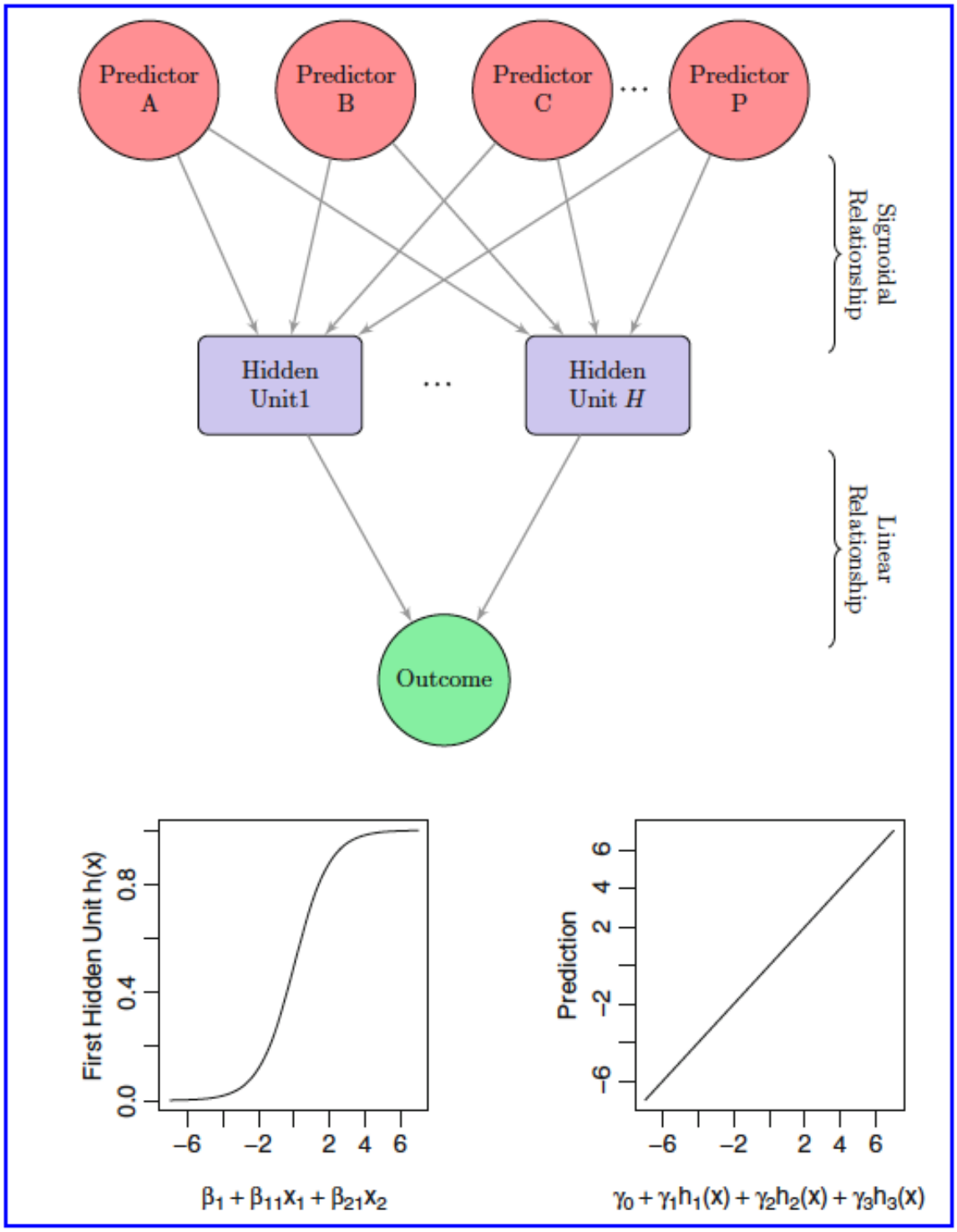


Fig. 7.1: A diagram of a neural network with a single hidden layer. The hidden units are linear combinations of the predictors that have been transformed by a sigmoidal function. The output is modeled by a linear combination of the hidden units

Figure 1: 1

As a consequence, only a subset of training set data points, where a /= 0, are needed for prediction. Since the regression line is determined using these samples, they are called the support vectors as they support the regression line.

kernel functions

Main tuning parameter: cost value

Centering and scaling the predictors prior to building an SVM model

7.4 K-Nearest Neighbours

To predict a new sample for regression, KNN identifies that sample's KNNs in the predictor space. The predicted response for the new sample is then the mean of the K neighbours' responses

Distance metrics: Euclidean distance, Minkowski distance

All predictors should be centered and scaled prior to performing KNN

Impute the missing values

Number of neighbours K can be determined by resampling

Computation complexity

The KNN method can have poor predictive performance when local predictor structure is not relevant to the response

7.5 Computing

```
library(AppliedPredictiveModeling)
data(solubility)
```

Neural Networks

```
library(nnet)
# nnetFit <- nnet(predictors, outcome, size = 5, decay = 0.01, linout = TRUE,
                # reduce the amount of printed output
                # trace = FALSE,
                # Expand the number of iterations to find parameter estimates
                # maxit = 500,
                # and the number of parameters used by the model
                # MaxNWts = 5 * (ncol(predictors)+1)+5+1)
```

```
# nnetAvg <- avNNet(predictors, outcome, size = 5, decay = 0.01,
                  # specify how many models to average
                  # repeats = 5, linout = TRUE,
                  # reduce the amount of printed output
                  # trace = FALSE,
                  # expand the number of iterations to find parameter estimates
                  # maxit = 500,
                  # and the number of parameters used by the model
                  # MaxNWts = 5 * (ncol(predictors) + 1) + 5 + 1)
```

```
# predict(nnetFit, newData)
# predict(nnetAvg, newData)
```

```
# The findCorrelation takes a correlation matrix and determines the column numbers that should be remov
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
tooHigh <- findCorrelation(cor(solTrainXtrans),cutoff = 0.75)
trainXnnet <- solTrainXtrans[,-tooHigh]
testXnnet <- solTestXtrans[,-tooHigh]
```

```
# create a specific candidate set of models to evaluate
nnetGrid <- expand.grid(.decay = c(0, 0.01, 0.1), .size = c(1:10),
                        # to use bagging instead of different random seeds
                        .bag = FALSE)

set.seed(100)
nnetTune <- train(solTrainXtrans, solTrainY, 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)
```

```
## Error in train.default(solTrainXtrans, solTrainY, method = "avNNet", tuneGrid = nnetGrid, : object '
```

```
nnetTune
```

```
## Error in eval(expr, envir, enclos): object 'nnetTune' not found
```

Multivariate Adaptive Regression Splines

```
library(earth)
```

```
## Loading required package: Formula
```

```
## Loading required package: plotmo
```

```
## Loading required package: plotrix
```

```
## Loading required package: TeachingDemos
```

```
marsFit <- earth(solTrainXtrans, solTrainY)
marsFit
```

```
## Selected 40 of 47 terms, and 31 of 228 predictors
## Termination condition: RSq changed by less than 0.001 at 47 terms
## Importance: NumNonHAtoms, SurfaceArea2, MolWeight, SurfaceArea1, ...
## Number of terms at each degree of interaction: 1 39 (additive model)
## GCV 0.3873018    RSS 309.672    GRSq 0.9076346    RSq 0.9221793
```

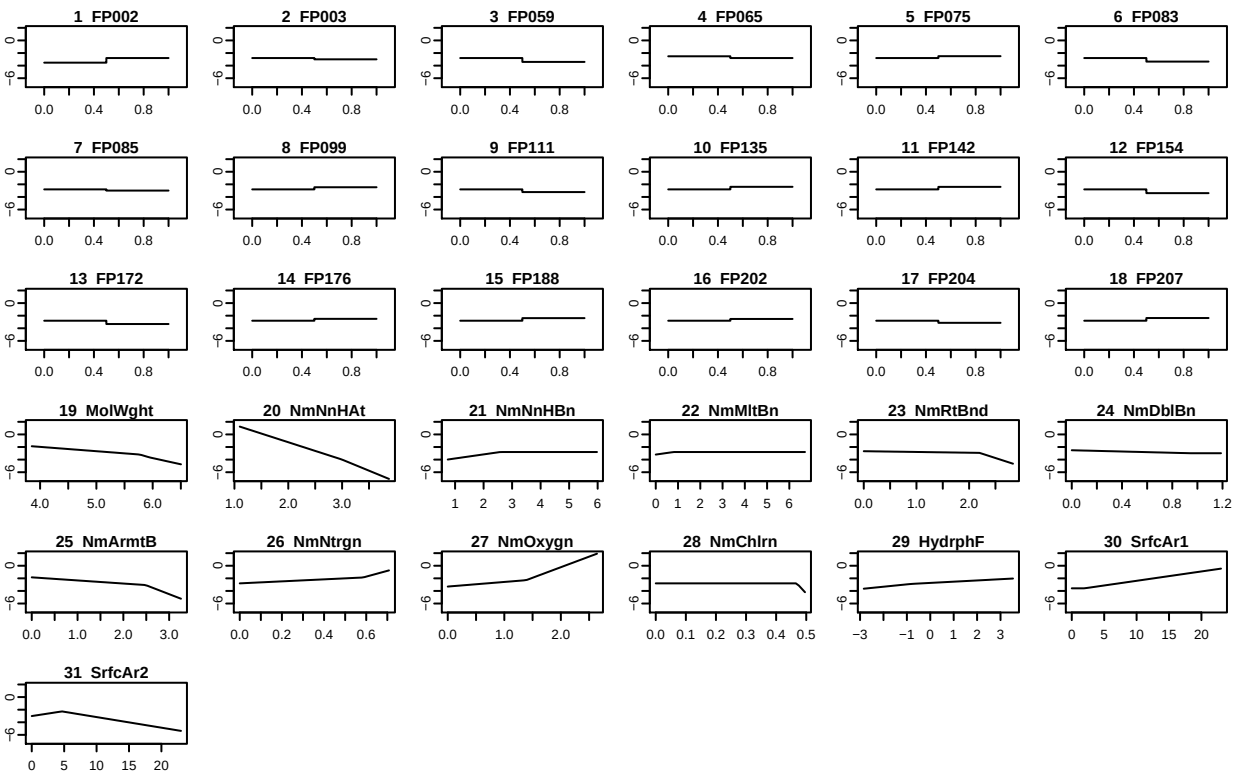
```
summary(marsFit)
```

```
## Call: earth(x=solTrainXtrans, y=solTrainY)
##
##                                  coefficients
## (Intercept)                         -4.455949
## FP002                                0.733904
## FP003                               -0.203502
## FP059                               -0.613495
## FP065                               -0.278610
## FP075                                0.295269
## FP083                               -0.563202
## FP085                               -0.193880
## FP099                                0.337591
## FP111                               -0.428246
## FP135                                0.405277
## FP142                                0.397119
## FP154                               -0.597650
## FP172                               -0.527362
## FP176                                0.294774
## FP188                                0.407757
## FP202                                0.279390
## FP204                               -0.336720
## FP207                                0.424005
## h(MolWeight-5.77157)                -1.964915
## h(5.94458-MolWeight)                 0.686131
## h(2.99573-NumNonHAtoms)              2.755144
## h(NumNonHAtoms-2.99573)             -3.517132
## h(2.57858-NumNonHBonds)             -0.647484
## h(0.79877-NumMultBonds)             -0.501790
## h(2.19722-NumRotBonds)               0.120247
## h(NumRotBonds-2.19722)              -2.709055
## h(0.941208-NumDblBonds)              0.501773
## h(2.48491-NumAromaticBonds)          0.492052
## h(NumAromaticBonds-2.48491)         -2.811877
## h(0.584815-NumNitrogen)             -1.593633
## h(NumNitrogen-0.584815)              9.162309
## h(1.38629-NumOxygen)                -0.730829
## h(NumOxygen-1.38629)                 3.358142
## h(NumChlorine-0.46875)             -51.880014
## h(-0.816625-HydrophilicFactor)      -0.374411
## h(HydrophilicFactor- -0.816625)      0.196559
## h(SurfaceArea1-1.9554)               0.148552
## h(4.66178-SurfaceArea2)             -0.158472
## h(SurfaceArea2-4.66178)             -0.168509
##
## Selected 40 of 47 terms, and 31 of 228 predictors
## Termination condition: RSq changed by less than 0.001 at 47 terms
```

```
## Importance: NumNonHAtoms, SurfaceArea2, MolWeight, SurfaceArea1, ...
## Number of terms at each degree of interaction: 1 39 (additive model)
## GCV 0.3873018    RSS 309.672    GRSq 0.9076346    RSq 0.9221793
```

```
plotmo(marsFit)
```

```
##  plotmo grid:    FP001 FP002 FP003 FP004 FP005 FP006 FP007 FP008 FP009
##                      0     1     0     1     1     0     0     0     0
##  FP010 FP011 FP012 FP013 FP014 FP015 FP016 FP017 FP018 FP019 FP020 FP021
##      0     0     0     0     0     1     0     0     0     0     0     0
##  FP022 FP023 FP024 FP025 FP026 FP027 FP028 FP029 FP030 FP031 FP032 FP033
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP034 FP035 FP036 FP037 FP038 FP039 FP040 FP041 FP042 FP043 FP044 FP045
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP046 FP047 FP048 FP049 FP050 FP051 FP052 FP053 FP054 FP055 FP056 FP057
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP058 FP059 FP060 FP061 FP062 FP063 FP064 FP065 FP066 FP067 FP068 FP069
##      0     0     0     0     0     0     0     1     1     0     0     0
##  FP070 FP071 FP072 FP073 FP074 FP075 FP076 FP077 FP078 FP079 FP080 FP081
##      0     0     1     0     0     0     0     0     0     1     0     0
##  FP082 FP083 FP084 FP085 FP086 FP087 FP088 FP089 FP090 FP091 FP092 FP093
##      1     0     0     0     0     1     0     0     0     0     0     0
##  FP094 FP095 FP096 FP097 FP098 FP099 FP100 FP101 FP102 FP103 FP104 FP105
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP106 FP107 FP108 FP109 FP110 FP111 FP112 FP113 FP114 FP115 FP116 FP117
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP118 FP119 FP120 FP121 FP122 FP123 FP124 FP125 FP126 FP127 FP128 FP129
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP130 FP131 FP132 FP133 FP134 FP135 FP136 FP137 FP138 FP139 FP140 FP141
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP142 FP143 FP144 FP145 FP146 FP147 FP148 FP149 FP150 FP151 FP152 FP153
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP154 FP155 FP156 FP157 FP158 FP159 FP160 FP161 FP162 FP163 FP164 FP165
##      0     0     0     0     0     0     0     0     0     0     1     0
##  FP166 FP167 FP168 FP169 FP170 FP171 FP172 FP173 FP174 FP175 FP176 FP177
##      0     0     1     0     0     0     0     0     0     0     0     0
##  FP178 FP179 FP180 FP181 FP182 FP183 FP184 FP185 FP186 FP187 FP188 FP189
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP190 FP191 FP192 FP193 FP194 FP195 FP196 FP197 FP198 FP199 FP200 FP201
##      0     0     0     0     0     0     0     0     0     0     0     0
##  FP202 FP203 FP204 FP205 FP206 FP207 FP208 MolWeight NumAtoms NumNonHAtoms
##      0     0     0     0     0     0     0  5.194234 3.135494     2.564949
##  NumBonds NumNonHBonds NumMultBonds NumRotBonds NumDblBonds
##  3.178054     3.351388     2.944766    1.098612   0.5670767
##  NumAromaticBonds NumHydrogen NumCarbon NumNitrogen NumOxygen NumSulfer
##           1.94591    3.691453  3.317541           0 0.6931472         0
##  NumChlorine NumHalogen  NumRings HydrophilicFactor SurfaceArea1
##            0          0 0.6931472        -0.3630242      7.25813
##  SurfaceArea2
##      7.759912
```




```
# Define the candidate models to test
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

# Fix the seed so that the results can be reproduced
set.seed(100)
marsTune <- train(solTrainXtrans, solTrainY, method = "earth",
                  # explicitly declare the candidate models to test
                  tuneGrid = marsGrid, trControl = trainControl(method = "cv"))
marsTune
```

```
## Multivariate Adaptive Regression Spline
##
## 951 samples
## 228 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 856, 856, 855, 855, 857, 856, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE       Rsquared   MAE
##   1        2      1.5610000  0.4198695  1.1947358
##   1        3      1.0734760  0.7254086  0.8215423
##   1        4      0.9865814  0.7648470  0.7598549
##   1        5      0.9107653  0.7992922  0.7060994
##   1        6      0.8843640  0.8119566  0.6828926
##   1        7      0.8504298  0.8276453  0.6625704
```

```
##   1        8      0.8394580  0.8324576  0.6511954
##   1        9      0.8476625  0.8290399  0.6542765
##   1       10      0.8369444  0.8322989  0.6466129
##   1       11      0.8247644  0.8365905  0.6370129
##   1       12      0.8220283  0.8385922  0.6330253
##   1       13      0.8019738  0.8454965  0.6148967
##   1       14      0.8011014  0.8463156  0.6130984
##   1       15      0.7771126  0.8542163  0.5927403
##   1       16      0.7751971  0.8552907  0.5874717
##   1       17      0.7611348  0.8611686  0.5754540
##   1       18      0.7641831  0.8598134  0.5761603
##   1       19      0.7568118  0.8626007  0.5753881
##   1       20      0.7510054  0.8638166  0.5700901
##   1       21      0.7401506  0.8681357  0.5632192
##   1       22      0.7381439  0.8691587  0.5603074
##   1       23      0.7323371  0.8717794  0.5563997
##   1       24      0.7298182  0.8730980  0.5529977
##   1       25      0.7251940  0.8747794  0.5498593
##   1       26      0.7206104  0.8765653  0.5473801
##   1       27      0.7131833  0.8788577  0.5417514
##   1       28      0.7106246  0.8798295  0.5394047
##   1       29      0.7040041  0.8823640  0.5367218
##   1       30      0.6974296  0.8847435  0.5312363
##   1       31      0.6932285  0.8859813  0.5275455
##   1       32      0.6931796  0.8859978  0.5282015
##   1       33      0.6926030  0.8862509  0.5289830
##   1       34      0.6952915  0.8852935  0.5298165
##   1       35      0.6945886  0.8853776  0.5313983
##   1       36      0.6928603  0.8858856  0.5298756
##   1       37      0.6940250  0.8855136  0.5297666
##   1       38      0.6949774  0.8853922  0.5286863
##   2        2      1.5294456  0.4428165  1.1620006
##   2        3      1.0824110  0.7182268  0.8327356
##   2        4      0.9635129  0.7755826  0.7419396
##   2        5      0.9326426  0.7919603  0.7207043
##   2        6      0.8876220  0.8130326  0.6933585
##   2        7      0.8568907  0.8263090  0.6654643
##   2        8      0.8346452  0.8348287  0.6488127
##   2        9      0.8198515  0.8402751  0.6420005
##   2       10      0.8108832  0.8446483  0.6268577
##   2       11      0.7900461  0.8535300  0.6043111
##   2       12      0.7757432  0.8576816  0.5926769
##   2       13      0.7470639  0.8673689  0.5720519
##   2       14      0.7462793  0.8682889  0.5653457
##   2       15      0.7432738  0.8694584  0.5621209
##   2       16      0.7389807  0.8711719  0.5589512
##   2       17      0.7259612  0.8748991  0.5528951
##   2       18      0.7161036  0.8783624  0.5434706
##   2       19      0.7040665  0.8821110  0.5360343
##   2       20      0.7016984  0.8830359  0.5363442
##   2       21      0.6978192  0.8842055  0.5317873
##   2       22      0.6924645  0.8856599  0.5277082
##   2       23      0.6834166  0.8887996  0.5229403
##   2       24      0.6772702  0.8905892  0.5195238
```

```
##   2       25      0.6768740  0.8905677  0.5170899
##   2       26      0.6810407  0.8893718  0.5205627
##   2       27      0.6803117  0.8893751  0.5188762
##   2       28      0.6781557  0.8900516  0.5176655
##   2       29      0.6798477  0.8895898  0.5177566
##   2       30      0.6830563  0.8886526  0.5183821
##   2       31      0.6824194  0.8887156  0.5165854
##   2       32      0.6816885  0.8889826  0.5157068
##   2       33      0.6815347  0.8890479  0.5146449
##   2       34      0.6821580  0.8888878  0.5129228
##   2       35      0.6776449  0.8903154  0.5076160
##   2       36      0.6776179  0.8903698  0.5075819
##   2       37      0.6767007  0.8906302  0.5071598
##   2       38      0.6768094  0.8906070  0.5072638
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 37 and degree = 2.
```

```
head(predict(marsTune, solTestXtrans))
```

```
##                y
## [1,]  0.05665338
## [2,]  0.22874960
## [3,] -0.51978335
## [4,] -0.17463927
## [5,] -0.35379766
## [6,]  0.68242010
```

```
varImp(marsTune)
```

```
## earth variable importance
##
##   only 20 most important variables shown (out of 228)
##
##                  Overall
## MolWeight        100.000
## NumNonHAtoms      92.084
## SurfaceArea1      92.084
## FP137             48.527
## NumHydrogen       38.121
## NumRotBonds       35.909
## NumOxygen         29.496
## FP043             23.949
## FP040             23.470
## SurfaceArea2      23.172
## FP101             19.746
## FP021             16.014
## NumMultBonds      16.014
## FP170             10.971
## FP059              9.569
## NumAromaticBonds   6.555
## NumHalogen         4.959
## FP070              0.000
```

```
## FP029              0.000
## FP147              0.000
```

Support Vector Machines

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
# svmFit <- ksvm(x = solTrainXtrans, y = solTrainY, kernel = "rbfdot", kpar = "automatic", C = 1, epsil
```

```
svmRTuned <- train(solTrainXtrans, solTrainY, method = "svmRadial", preProc = c("center","scale"),
                   tuneLength = 14, trControl = trainControl(method = "cv"))
svmRTuned
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 951 samples
## 228 predictors
##
## Pre-processing: centered (228), scaled (228)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 857, 856, 856, 856, 855, 856, ...
## Resampling results across tuning parameters:
##
##   C        RMSE       Rsquared   MAE
##      0.25  0.8086472  0.8650491  0.6040350
##      0.50  0.7158482  0.8871585  0.5331541
##      1.00  0.6684252  0.8978323  0.4960238
##      2.00  0.6366349  0.9052559  0.4700727
##      4.00  0.6217857  0.9088492  0.4571951
##      8.00  0.6059898  0.9132655  0.4473604
##     16.00  0.6009921  0.9145500  0.4425787
##     32.00  0.6017992  0.9142121  0.4429745
##     64.00  0.6011190  0.9142660  0.4441073
##    128.00  0.6017550  0.9139535  0.4448852
##    256.00  0.6038782  0.9133373  0.4473942
##    512.00  0.6055245  0.9127945  0.4489103
##   1024.00  0.6065738  0.9124243  0.4498322
##   2048.00  0.6090538  0.9116155  0.4528802
##
## Tuning parameter 'sigma' was held constant at a value of 0.002662972
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.002662972 and C = 16.
```

```
svmRTuned$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 16
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0026629721069908
##
## Number of Support Vectors : 623
##
## Objective Function Value : -303.8084
## Training error : 0.01011
```

K-Nearest Neighbors

```
# remove a few sparse and unbalanced fingerprints first
knnDescr <- solTrainXtrans[,-nearZeroVar(solTrainXtrans)]
set.seed(100)
knnTune <- train(knnDescr, solTrainY, method = "knn",
                 # center and scaling will occur for new predictions too
                 preProc = c("center", "scale"),
                 tuneGrid = data.frame(.k=1:20),
                 trControl = trainControl(method = "cv"))
knnTune
```

```
## k-Nearest Neighbors
##
## 951 samples
## 225 predictors
##
## Pre-processing: centered (225), scaled (225)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 856, 856, 855, 855, 857, 856, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    1  1.250232  0.6611312  0.9165221
##    2  1.103952  0.7197662  0.7988127
##    3  1.065688  0.7350824  0.7903996
##    4  1.043534  0.7449116  0.7860586
##    5  1.049529  0.7409384  0.7889923
##    6  1.058417  0.7359120  0.7986353
##    7  1.060938  0.7331595  0.7980003
##    8  1.055874  0.7358010  0.7974424
##    9  1.065884  0.7311823  0.8082185
##   10  1.070847  0.7286249  0.8134830
##   11  1.069596  0.7302802  0.8124852
##   12  1.076953  0.7269775  0.8202945
##   13  1.088453  0.7204797  0.8318907
```

```
##    14  1.093359  0.7181335  0.8380720
##    15  1.100073  0.7149682  0.8461514
##    16  1.104751  0.7127331  0.8533856
##    17  1.109370  0.7101825  0.8581988
##    18  1.119957  0.7059283  0.8669092
##    19  1.127248  0.7024141  0.8742935
##    20  1.132476  0.6998827  0.8761286
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 4.
```

```
plot(knnTune)
```