Exercises for
Applied Predictive Modeling
Chapter 7 — Nonlinear Regression Models

Max Kuhn, Kjell Johnson

Version 1
January 8, 2015

The solutions in this file uses several R packages not used in the text. To install all of the packages needed for this document, use:

```
> install.packages(c("AppliedPredictiveModeling", "caret", "earth", "kernlab",
+                    "latticeExtra", "mlbench", "nnet", "plotmo"))
```

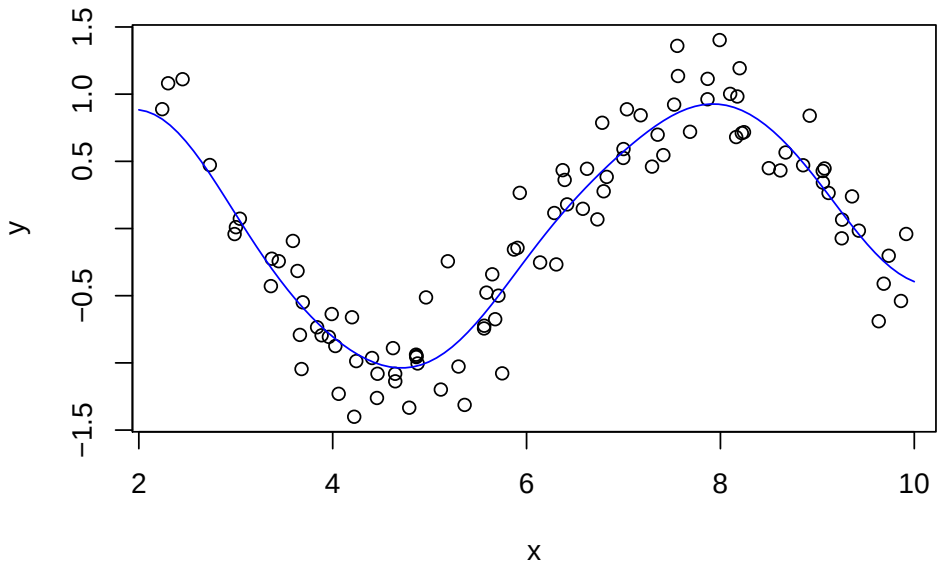
Exercise 1

Investigate the relationship between the cost, ϵ and kernel parameters for a support vector machine model. Simulate a single predictor and a non–linear relationship, such as a *sin* wave shown in Fig. 7.7:

```
> set.seed(100)
> x <- runif(100, min = 2, max = 10)
> y <-  sin(x) + rnorm(length(x)) * .25
> sinData <- data.frame(x = x, y = y)
>
> ## Create a grid of x values to use for prediction
> dataGrid <- data.frame(x = seq(2, 10, length = 100))
```

(a) Fit different models using a radial basis function and different values of the cost (the `C` parameter) and ϵ. Plot the fitted curve. For example:

```
> library(kernlab)
> rbfSVM <- ksvm(x = x, y = y, data = sinData,
+                kernel ="rbfdot", kpar = "automatic",
+                C = 1, epsilon = 0.1)
> modelPrediction <- predict(rbfSVM, newdata = dataGrid)
> ## This is a matrix with one column. We can plot the
> ## model predictions by adding points to the previous plot
> plot(x, y)
> points(x = dataGrid$x, y = modelPrediction[,1],
+        type = "l", col = "blue")
```




```
> ## Try other parameters
```

(b) The σ parameter can be adjusted using the `kpar` argument, such as `kpar = list(sigma = 1)`. Try different values of σ to understand how this parameter changes the model fit. How do the cost, ϵ and σ values affect the model?

Solutions

A series of SVM models will be fit over four values of ϵ and the cost parameter. A fixed value of σ determined automatically using the `sigest` function in the kernlab package. The same value is guaranteed by setting the random number seed before each SVM model.

```
> svmParam1 <- expand.grid(eps = c(.01, 0.05, .1, .5), costs = 2^c(-2, 0, 2, 8))
>
> for(i in 1:nrow(svmParam1)) {
+   set.seed(131)
+   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("cost:", format(svmParam1$costs)[i]))
+   svmPred1 <- if(i == 1) tmp else rbind(tmp, svmPred1)
+ }
> svmPred1$costs <- factor(svmPred1$costs, levels = rev(levels(svmPred1$costs)))
```

Figure 1 shows the training data and the regression curve for each model fit. Generally speaking, the regression model becomes more jagged as the cost value increases. This is due to the training process putting a cost on the residual values. As a result, higher cost values will motivate the training process to minimize the residuals on the training set data as much as possible. This leads to models with lower bias and higher variability (i.e. the fit is likely to change if the data changes). The effect of ϵ is similar but not has potent. As ϵ *decreases* the model fit begins to overfit more. This effect is smaller than the cost effect. For example, in the left–most column, the amount of overfitting is negligible. However, in the right–most column, the overfitting effect can be seen the most at the extremes of the x–axis where the model fit changes more radically as ϵ becomes small. Next, the effect of σ is factored into the analysis:

```
> set.seed(1016)
> svmParam2 <- expand.grid(eps = c(.01, 0.05, .1, .5), costs = 2^c(-2, 0, 2, 8),
+                          sigma = as.vector(sigest(y~x, data = sinData, frac = .75)))
>
> for(i in 1:nrow(svmParam2)) {
+   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$eps)[i]),
+                     costs = paste("cost:", svmParam2$costs[i]),
+                     sigma = paste("sigma:", format(svmParam2$sigma, digits = 2)[i]))
+   svmPred2 <- if(i == 1) tmp else rbind(tmp, svmPred2)
+ }
> svmPred2$costs <- factor(svmPred2$costs, levels = rev(levels(svmPred2$costs)))
> svmPred2$sigma <- factor(svmPred2$sigma, levels = rev(levels(svmPred2$sigma)))
```

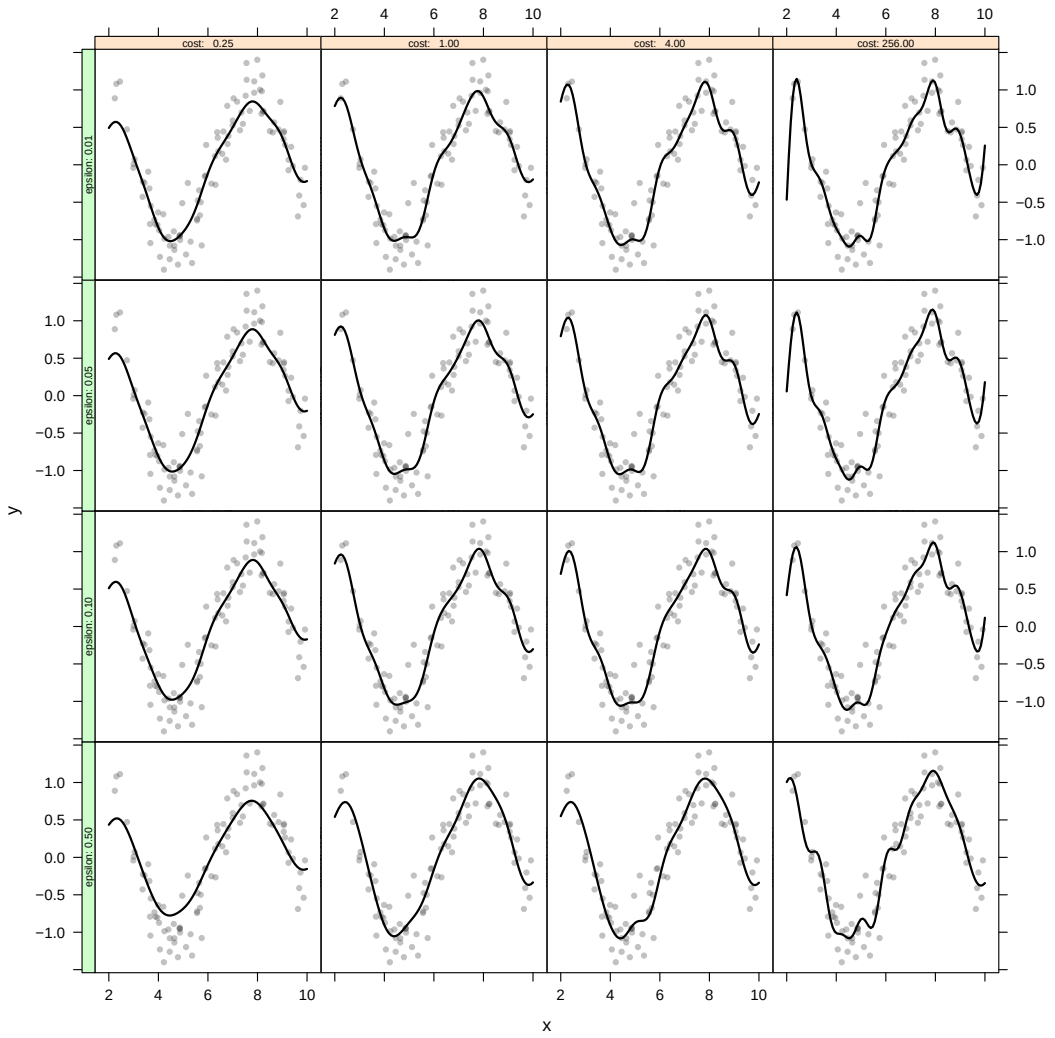


Figure 1: The relationship between the costs values and ϵ for the simulated data.

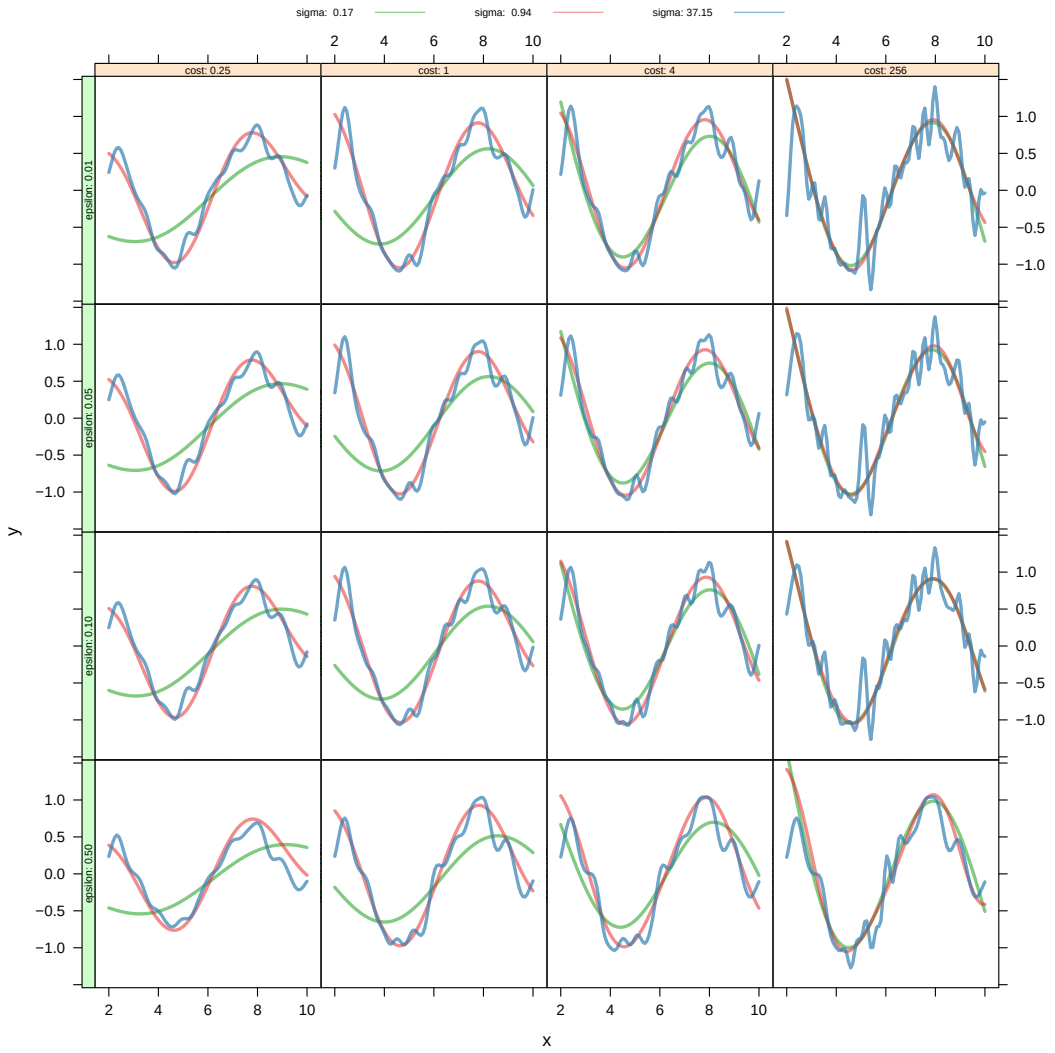


Figure 2: The relationship between, σ, the costs values and ϵ for the simulated data

Figure 2 shows the fitted regression curves. The effect of σ is complex. For low to moderate cost values, σ appears to effect the model bias. For example, in the left column (i.e. low cost) the curves are dynamic in σ; the low value tends to underfit the data, while high value tends to overfit the data. As cost increases, the low and mid value of σ are similar, while the high value still tends to overfit.

This low-dimensional example illustrates the powerful fitting ability that SVMs have through the tuning parameters. Clearly SVMs are prone to overfitting and must be appropriately trained/tuned to protect against overfitting to the training data.

Exercise 2

? introduced several benchmark datasets create by simulation. One of these simulations used the following non–linear equation to create data:

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + N(0, \sigma^2)$$

where the x values are random variables uniformly distributed between [0, 1] (there are also 5 other non–informative variables also created in the simulation). The package mlbench contains a function called `mlbench.friedman1` that can simulate these data:

```
> library(mlbench)
> set.seed(200)
> trainingData <- mlbench.friedman1(200, sd = 1)
> ## We convert the 'x' data from a matrix to a data frame
> ## One reason is that this will give the columns names.
> trainingData$x <- data.frame(trainingData$x)
>
> ## Look at the data using
> ## 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)
```

Tune several models on these data. For example:

```
> library(caret)
> set.seed(921)
> knnModel <- train(x = trainingData$x,
+                   y = trainingData$y,
+                   method = "knn",
+                   preProc = c("center", "scale"),
+                   tuneLength = 10)
> knnModel


k-Nearest Neighbors

200 samples
 10 predictor

Pre-processing: centered, scaled
Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
```

```
Resampling results across tuning parameters:

  k   RMSE   Rsquared  RMSE SD  Rsquared SD
   5  3.489  0.5020    0.2659   0.07413
   7  3.325  0.5485    0.2275   0.06433
   9  3.225  0.5854    0.2426   0.06904
  11  3.178  0.6092    0.2594   0.07305
  13  3.184  0.6184    0.2524   0.06996
  15  3.188  0.6251    0.2409   0.06605
  17  3.214  0.6282    0.1891   0.05315
  19  3.209  0.6404    0.1922   0.05056
  21  3.215  0.6487    0.1831   0.04961
  23  3.235  0.6528    0.1870   0.04752

RMSE was used to select the optimal model using  the smallest value.
The final value used for the model was k = 11.
```

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

    RMSE Rsquared
   3.122    0.669
```

Which models appear to give the best performance? Does MARS select the informative predictors (those named `X1` – `X5`)?

Solutions

K-nearest neighbors models have better performance when the underlying relationship between predictors and the response relies is dependent on samples' proximity in the predictor space. Geographic information is not part of the data generation scheme for this particular data set. Hence, we would expect another type of model to perform better then K-NN. Let's try MARS and SVM radial basis function models.

```
> marsGrid <- expand.grid(degree = 1:2, nprune = seq(2,14,by=2))
> set.seed(921)
> marsModel <- train(x = trainingData$x,
+                    y = trainingData$y,
+                    method = "earth",
+                    preProc = c("center", "scale"),
+                    tuneGrid = marsGrid)
>
> marsPred <- predict(marsModel, newdata = testData$x)
```

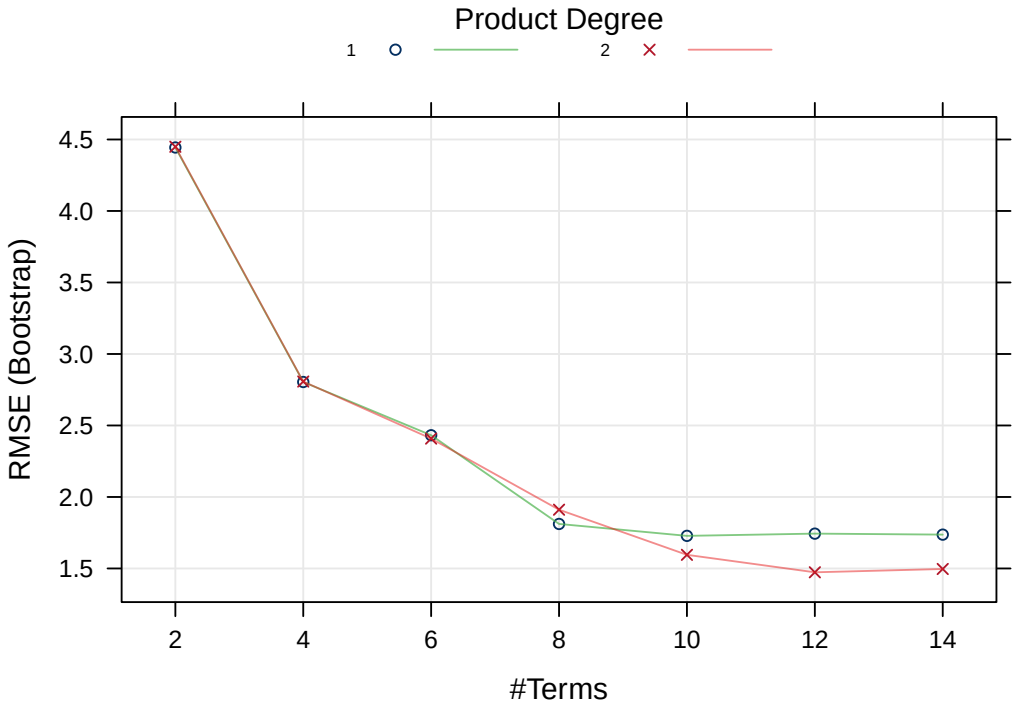


Figure 3: The tuning parameter profile for the MARS model.

```
>
> postResample(pred = marsPred, obs = testData$y)

    RMSE Rsquared
  1.2656   0.9351
```

Figure 3 illustrates the MARS tuning parameter profile. The optimal model in this case uses 12 terms, has degree 2, with an RMSE of 1.47%.

There are a few ways we can decipher which predictors were used in the final model. One way to do this is by using the variable importance scores:

```
> varImp(marsModel)

earth variable importance

    Overall
X1    100.0
X4     85.0
X2     68.9
```

```
X5     48.5
X3     38.9
X9      0.0
X8      0.0
X6      0.0
X7      0.0
X10     0.0
```

Another way would be to generate a summary of the model which gives the exact form. We will re-create the model using the `earth` function directly:

```
> marsFit <- earth(x = trainingData$x,
+                  y = trainingData$y,
+                  nprune = 12, degree = 2)
> summary(marsFit)


Call: earth(x=trainingData$x, y=trainingData$y, nprune=12, degree=2)

                                coefficients
(Intercept)                           21.784
h(0.637977-X1)                       -14.869
h(X1-0.637977)                        12.168
h(0.593908-X2)                       -20.237
h(0.496556-X3)                        10.559
h(X3-0.496556)                        10.849
h(0.759836-X4)                        -9.863
h(X4-0.759836)                        10.598
h(0.86944-X5)                         -5.503
h(X1-0.637977) * h(X2-0.295997)      -48.147
h(0.701018-X1) * h(0.593908-X2)       26.434

Selected 11 of 20 terms, and 5 of 10 predictors
Termination condition: Reached nk 21
Importance: X1, X4, X2, X5, X3, X6-unused, X7-unused, X8-unused, X9-unused, ...
Number of terms at each degree of interaction: 1 8 2
GCV 1.745    RSS 264.1    GRSq 0.9292    RSq 0.9458
```

Clearly, MARS only selects `X1` – `X5` as important predictors in relationship to the response. Figure 4 uses the plotmo package to create plots of each predictor versus the outcome.

```
> set.seed(921)
> svmRModel <- train(x = trainingData$x,
+                    y = trainingData$y,
+                    method = "svmRadial",
+                    preProc = c("center", "scale"),
+                    tuneLength = 8)
>
> svmRPred <- predict(svmRModel, newdata = testData$x)
```

```
grid:    X1     X2     X3     X4     X5     X6     X7    X8     X9    X10
     0.5139 0.5107 0.5373 0.4446 0.5343 0.4976 0.4688 0.498 0.5289 0.5359
```

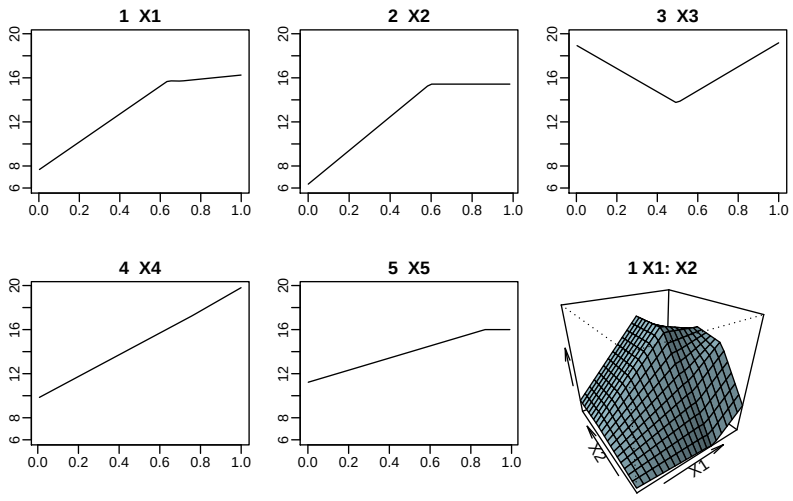


Figure 4: The functional form of the MARS model for the Friedman simulation data.

```
>
> postResample(pred = svmRPred, obs = testData$y)

    RMSE Rsquared
  2.0668   0.8268
```

Figure 5 illustrates the radial basis function SVM tuning parameter profile. The optimal model has a cost value of 16, and sigma value of 0.0605, with an RMSE of 2.05%.

Overall, the MARS model performs best, with the radial basis function SVM coming in next in performance. K-NN has relatively poor performance for this problem, which is not surprising since the data was not generated using neighborhood information.

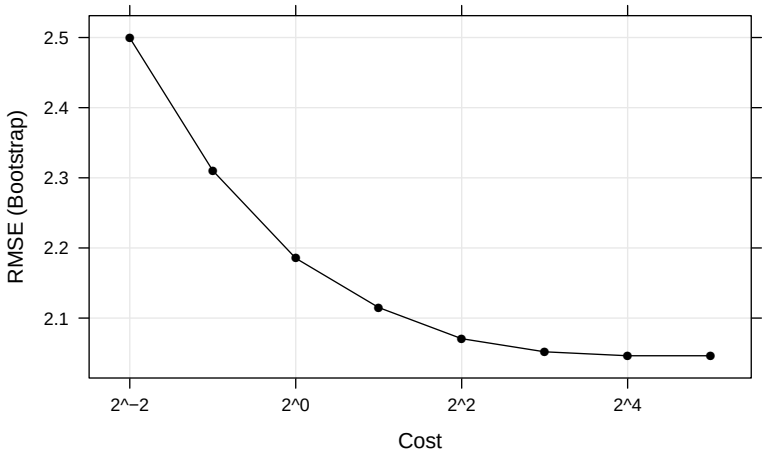


Figure 5: The tuning parameter profile for the SVM radial basis function model.

Exercise 3

For the Tecator data described in the last chapter, build SVM, neural network, MARS and K–nearest neighbor models. Since neural networks are especially sensitive to highly correlated predictors, does pre–processing using PCA help the model?

Solutions

The same data–splitting and resampling methodology from the last set of exercises was used:

```
> library(caret)
> data(tecator)
>
> set.seed(1029)
> inMeatTraining <- createDataPartition(endpoints[, 3], p = 3/4, list= FALSE)
>
> absorpTrain <- absorp[ inMeatTraining,]
> absorpTest  <- absorp[-inMeatTraining,]
> proteinTrain <- endpoints[ inMeatTraining, 3]
> proteinTest  <- endpoints[-inMeatTraining,3]
>
> ctrl <- trainControl(method = "repeatedcv", repeats = 5)
```

A MARS model was fit to the data:

```
> set.seed(529)
> meatMARS <- train(x = absorpTrain, y = proteinTrain,
+                   method = "earth",
+                   trControl = ctrl,
+                   tuneLength = 25)
```

Figure 6 shows the results. In the end, 11 terms were retained (using 5 unique predictors) for a model with an associated RMSE of 1.12%. This is worse performance than the linear models form the last set of exercises. Would bagging the MARS model help? The bagging results are also presented in Figure 6. For this data, provides only a slight improvement over the original MARS model.

```
> set.seed(529)
> meatBMARS <- train(x = absorpTrain, y = proteinTrain,
+                    method = "bagEarth",
+                    trControl = ctrl,
+                    tuneLength = 25,
+                    B = 20)
>
> plotDat <- rbind(meatMARS$results, meatBMARS$results)
> plotDat$Model <- rep(c("Basic", "Bagged"), each = nrow(meatMARS$results))
```

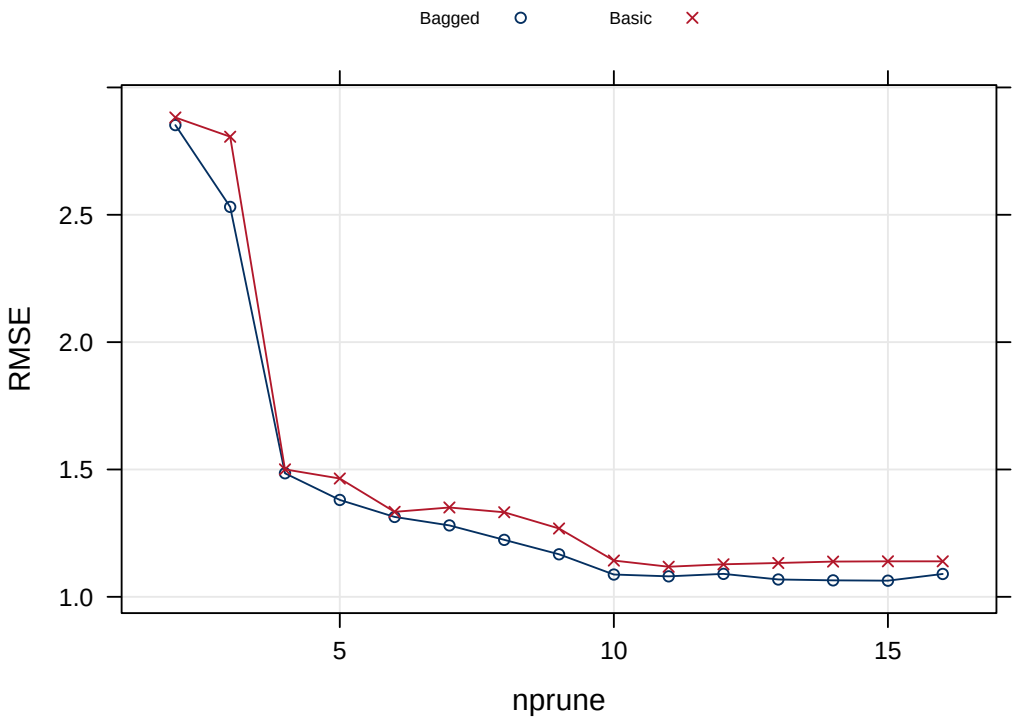


Figure 6: Resampled RMSE values versus the number of retained terms for the basic MARS model and a bagged version.

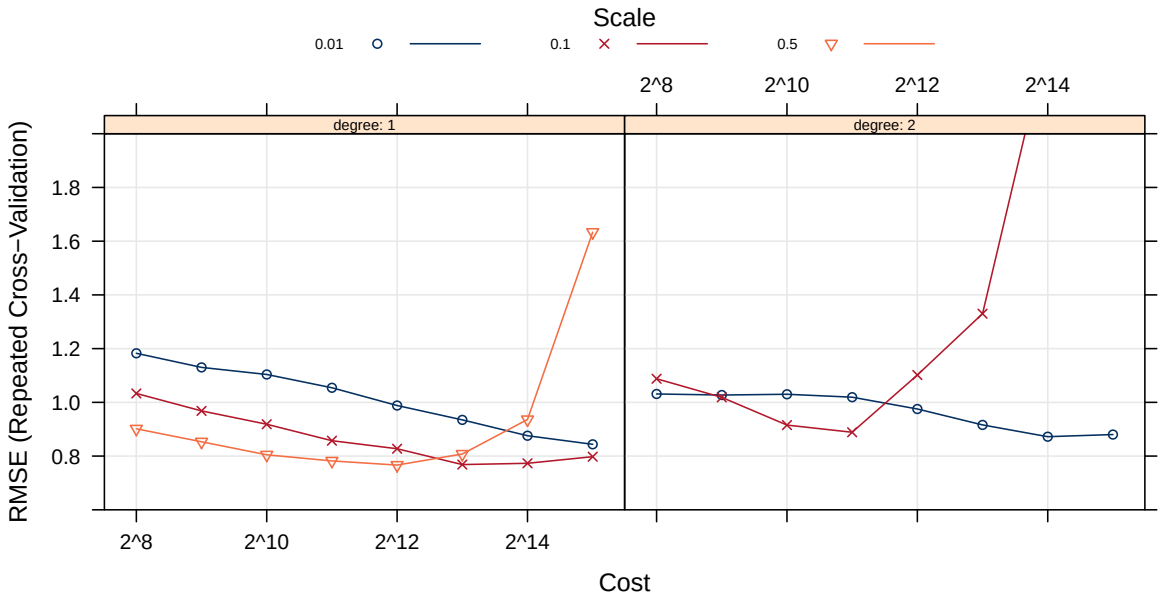


Figure 7: The RMSE resampling profiles for linear and quadratic support vector machines.

```
> polyGrid <- expand.grid(degree = 1:2,
+                         C = 2^seq(8, 15, length = 8),
+                         scale = c(.5, .1, 0.01))
> polyGrid <- polyGrid[!(polyGrid$scale == .5 & polyGrid$degree == 2),]
>
> set.seed(529)
> meatQSVM <- train(x = absorpTrain, y = proteinTrain,
+                   method = "svmPoly",
+                   preProcess = c("center", "scale"),
+                   trControl = ctrl,
+                   tuneGrid = polyGrid)
```

The next model examines the performance of a polynomial kernel support vector machine tuning over the parameters of degree, cost and scale. Figure 7 illustrates the results, where the optimal model has degree 1, a cost of 4096, and scale of 0.5. The RMSE of the optimal model is 0.77% which is better than the MARS models.

```
> set.seed(529)
> meatRSVM <- train(x = absorpTrain, y = proteinTrain,
+                   method = "svmRadial",
+                   preProcess = c("center", "scale"),
+                   trControl = ctrl,
+                   tuneLength = 10)
```

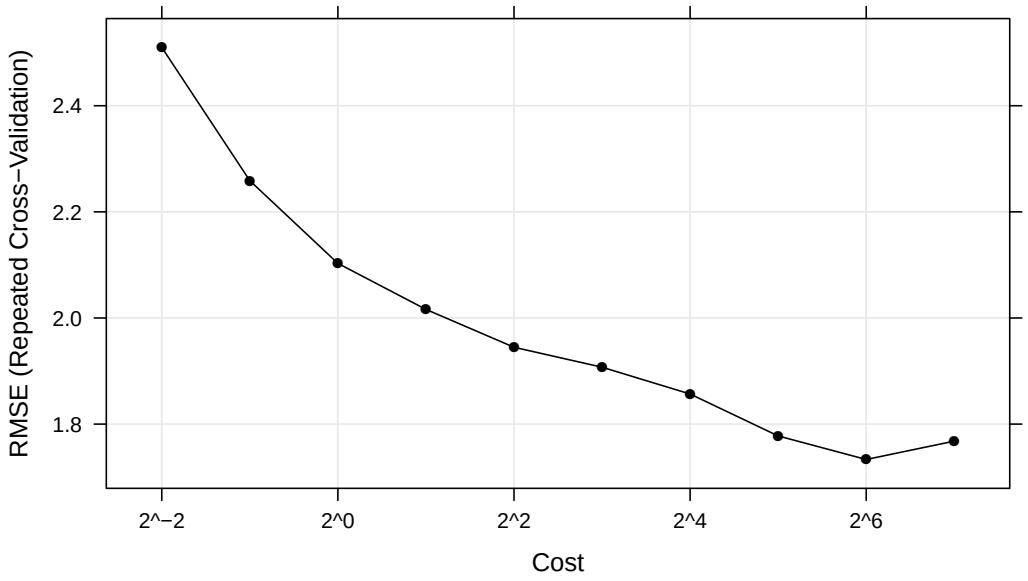


Figure 8: The relationship between the resampled RMSE values and the cost parameter for the RBF support vector machine.

To understand the performance of a radial basis support vector machine, we will search over 10 values of the cost parameter. Figure 8 illustrates the results, where the optimal model has a cost of 64. The RMSE of the optimal model is 1.73% which is worse than the polynomial SVM and MARS models.

For neural networks, we will tune over several values of the size and decay tuning parameters. In the first model, we will center and scale the predictors. In the second model, we will pre-process the predictors by centering, scaling, and using PCA. Figure 9 compares the tuning parameter profiles with and without PCA pre-processing. Notice that the RMSE is much higher for the PCA pre-processed set, indicating that PCA does not reduce the dimension of the predictor space in a way that is helpful for predicting absorption.

```
> set.seed(529)
> meatNet <- train(x = absorpTrain, y = proteinTrain,
+                  method = "nnet",
+                  trControl = ctrl,
+                  preProc = c("center", "scale"),
+                  linout = TRUE,
+                  trace = FALSE,
+                  tuneGrid = expand.grid(size = 1:9,
+                                         decay = c(0, .001, .01, .1)))
```

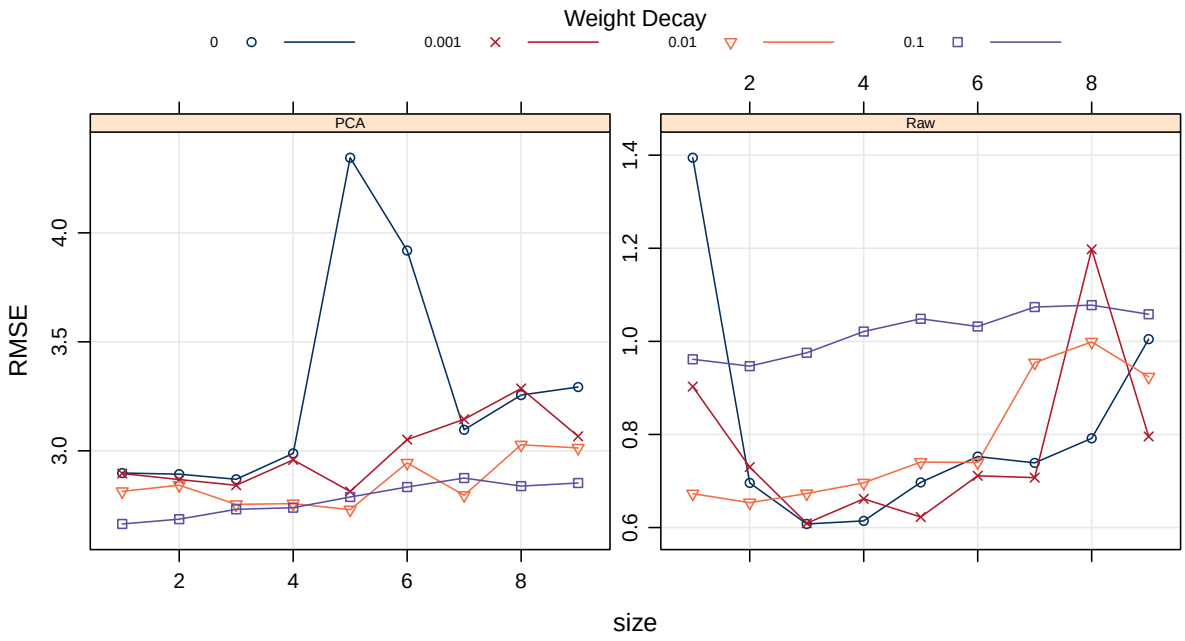


Figure 9: Neural network resampling profiles for the RMSE statistic for different pre–processing techniques: PCA signal extraction with centering and scaling versus centering and scaling alone (Raw).

```
> set.seed(529)
> meatPCANet <- train(x = absorpTrain, y = proteinTrain,
+                     method = "nnet",
+                     trControl = ctrl,
+                     preProc = c("center", "scale", "pca"),
+                     linout = TRUE,
+                     trace = FALSE,
+                     tuneGrid = expand.grid(size = 1:9,
+                                            decay = c(0, .001, .01, .1)))
> plotNNet <- rbind(meatNet$results, meatPCANet$results)
> plotNNet$Model <- rep(c("Raw", "PCA"), each = nrow(meatNet$results))
```

```
> set.seed(529)
> meatKnn <- train(x = absorpTrain, y = proteinTrain,
+                  method = "knn",
+                  trControl = ctrl,
+                  preProc = c("center", "scale"),
+                  tuneGrid = data.frame(.k = seq(1, 20, by = 2)))
```

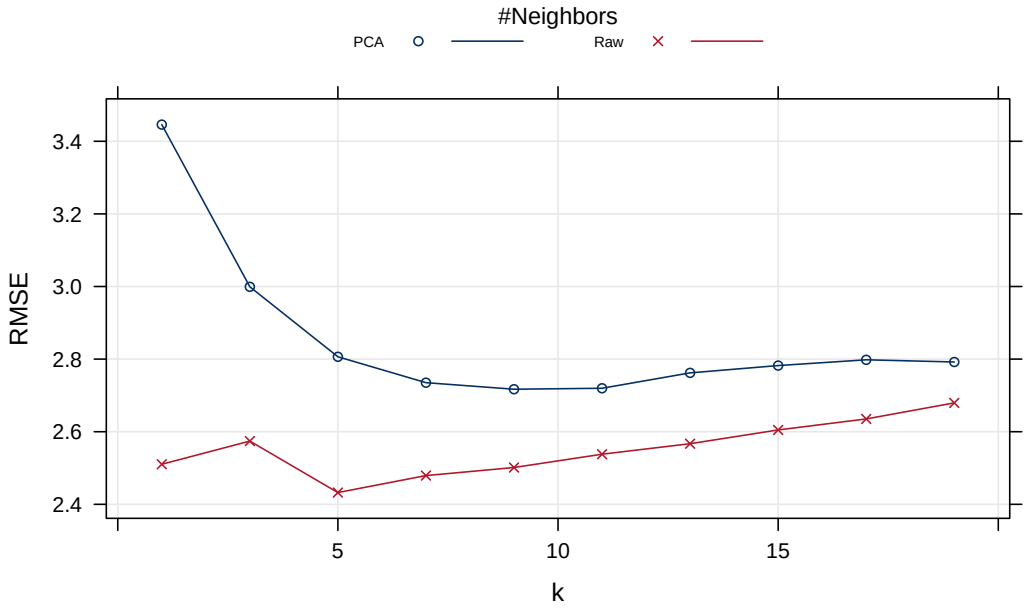


Figure 10: The resampling RMSE profile of the K–nearest neighbors models (with and without PCA pre–processing).

```
> set.seed(529)
> meatPCAKnn <- train(x = absorpTrain, y = proteinTrain,
+                     method = "knn",
+                     trControl = ctrl,
+                     preProc = c("center", "scale", "pca"),
+                     tuneGrid = data.frame(k = seq(1, 20, by = 2)))
>
> plotKnn <- rbind(meatKnn$results, meatPCAKnn$results)
> plotKnn$Model <- rep(c("Raw", "PCA"), each = nrow(meatKnn$results))
```

Last, a K-nearest neighbors model is built over several values of k where the predictors are pre-processed using centering and scaling, and are also pre-processed using centering, scaling, and PCA (Figure 10). Like neural networks, PCA does not provide useful dimension reduction for predicting the absorption response. Instead, it is better to pre-process by centering and scaling. Overall, the K-NN model performs the worst where RMSE of the optimal model is 2.43%.

```
> load("meatPLS.RData")
> meatResamples <- resamples(list(PLS = meatPLS,
+                                 MARS = meatBMARS,
+                                 SVMlin = meatQSVM,
+                                 SVMrad = meatRSVM,
+                                 NNet = meatNet,
```

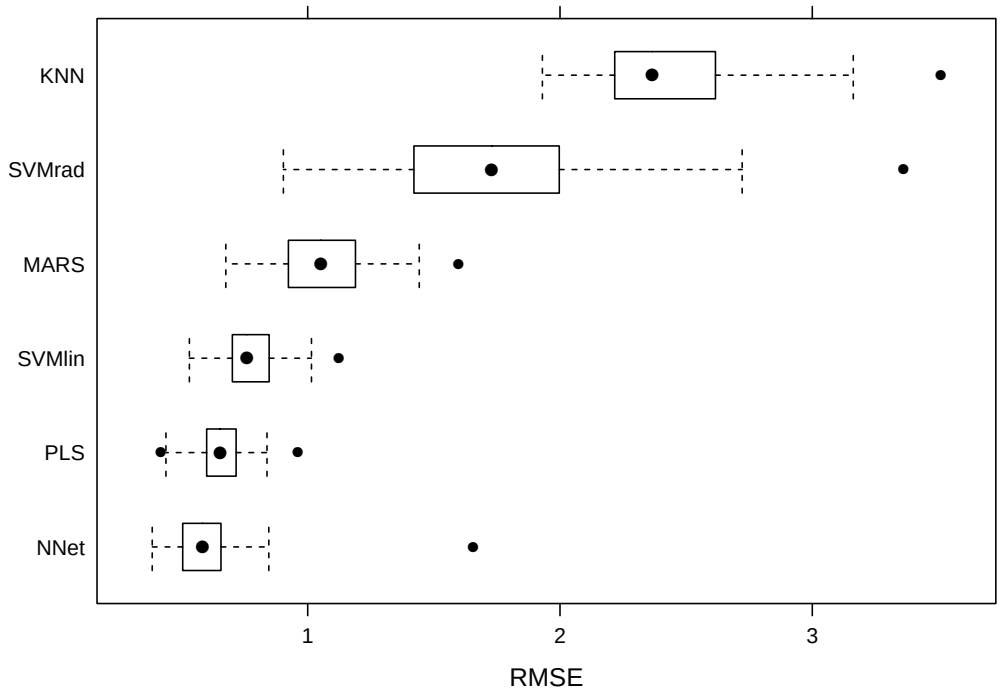


Figure 11: Resampling distributions of the RMSE statistic for the models fit in this document as well as the best form the linear models (i.e. PLS).

```
+                                KNN = meatKnn))
```

To compare models, the resampling distributions are presented in Figure 11. For this data, neural networks (centering and scaling, only) and partial least squares (also centering and scaling) have the lowest overall RMSE.

Exercise 4

Return to the permeability problem outlined in Exercise 6.2. Train several non–linear regression models and evaluate the resampling and test set performance.

(a) Which non–linear regression model gives the optimal resampling and test set performance?

(b) Do any of the non–linear models outperform the optimal linear model you previously developed in Exercise 6.2? If so, what might this tell you about the underlying relationship between the predictors and the response?

(c) Would you recommend any of the models you have developed to replace the permeability laboratory experiment?

Solutions

In order to make a parallel comparison to the results in Exercise 6.2, we need to perform the same pre-precessing steps and set up the identical validation approach. The following syntax provides the same pre-processing, data partition into training and testing sets, and validation set-up.

```
> library(AppliedPredictiveModeling)
> data(permeability)
>
> #Identify and remove NZV predictors
> nzvFingerprints = nearZeroVar(fingerprints)
> noNzvFingerprints <- fingerprints[,-nzvFingerprints]
>
> #Split data into training and test sets
> set.seed(614)
> trainingRows <- createDataPartition(permeability,
+                                     p = 0.75,
+                                     list = FALSE)
>
> trainFingerprints <- noNzvFingerprints[trainingRows,]
> trainPermeability <- permeability[trainingRows,]
>
> testFingerprints <- noNzvFingerprints[-trainingRows,]
> testPermeability <- permeability[-trainingRows,]
>
> set.seed(614)
> ctrl <- trainControl(method = "LGOCV")
```

Next, we will find optimal tuning parameters for MARS, SVM (radial basis function), and K-NN models.

```
> set.seed(614)
>
> marsPermGrid <- expand.grid(degree = 1:2, nprune = seq(2,14,by=2))
> marsPermTune <- train(x = trainFingerprints, y = log10(trainPermeability),
+                       method = "earth",
+                       tuneGrid = marsPermGrid,
+                       trControl = ctrl)
>
>
> RSVMPermGrid <- expand.grid(sigma = c(0.0005,0.001,0.0015),
+                             C = seq(1,49,by=6))
> RSVMPermTune <- train(x = trainFingerprints, y = log10(trainPermeability),
+                       method = "svmRadial",
+                       trControl = ctrl,
+                       tuneGrid = RSVMPermGrid)
>
> knnPermTune <- train(x = trainFingerprints, y = log10(trainPermeability),
+                      method = "knn",
+                      tuneLength = 10)
```

Figure 12 indicates that the optimal degree and number of terms that maximize R^2 are 1 and 8, respectively, with an R^2 of 0.49.

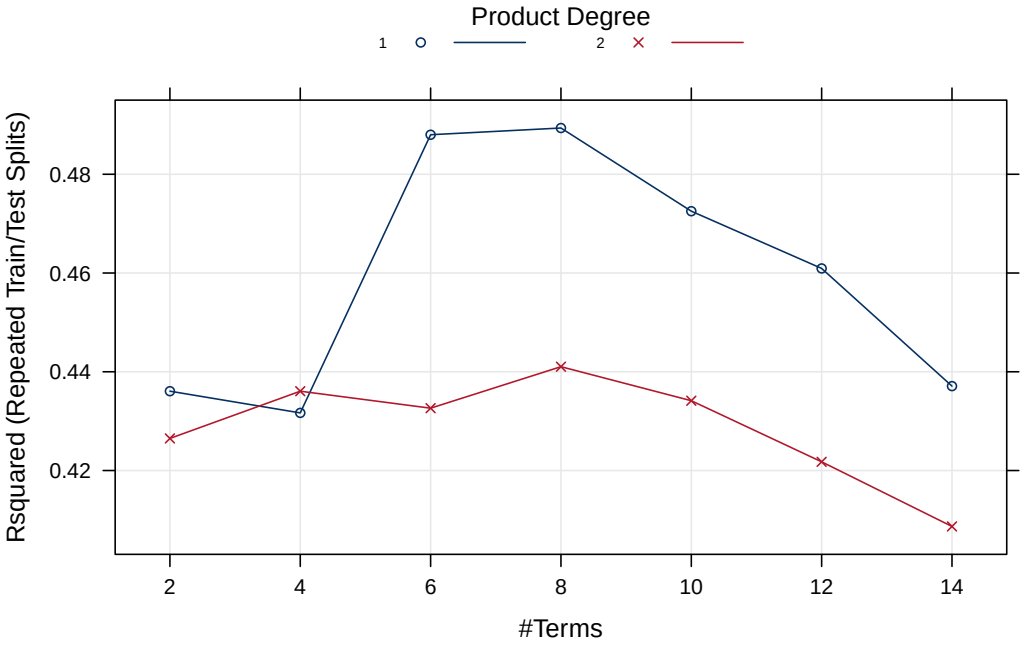


Figure 12: MARS tuning parameter profile for the permeability data

Figure 13 indicates that the optimal cost and sigma that maximize R^2 are 19 and 0.001, respectively,

with an R^2 of 0.55.

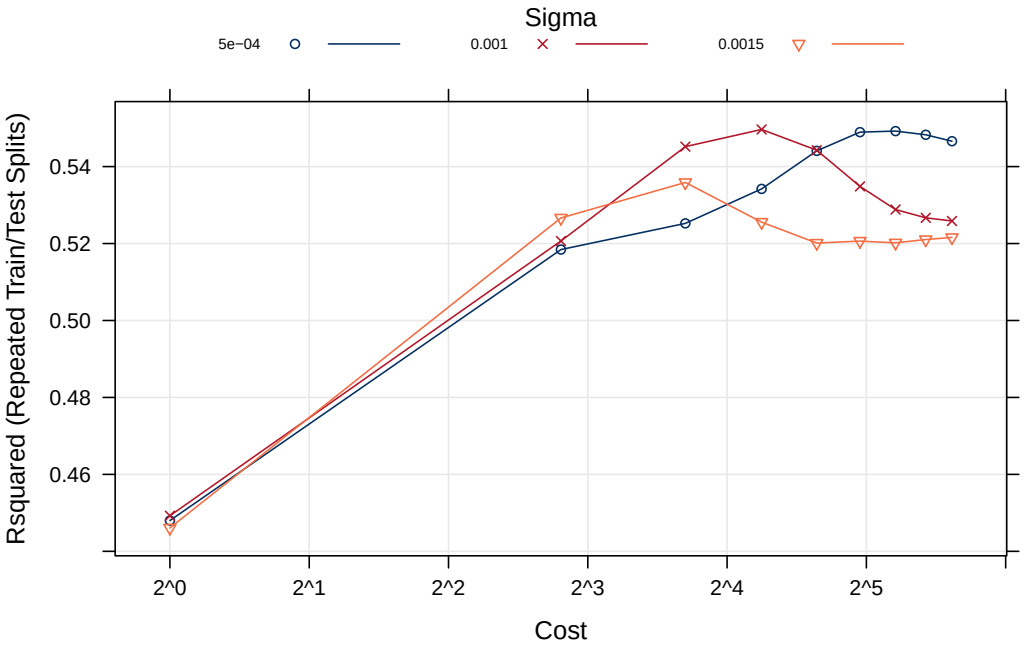


Figure 13: Radial basis function SVM tuning parameter profile for the permeability data

Figure 14 indicates that the optimal number of nearest neighbors that maximize R^2 are 5, respectively, with an R^2 of 0.25.

For these three non-linear models, the radial basis function SVM model performs best with a leave-group out cross-validated R^2 value of 0.55. This is worse than the elastic net model tuned in Exercise 6.2 which had a leave-group out cross-validated R^2 value of 0.58. These results indicate that the underlying relationship between the predictors and the response is likely best described by a linear structure in a reduced dimension of the original space.

For the models tuned thus far, we would recommend the elastic net model.

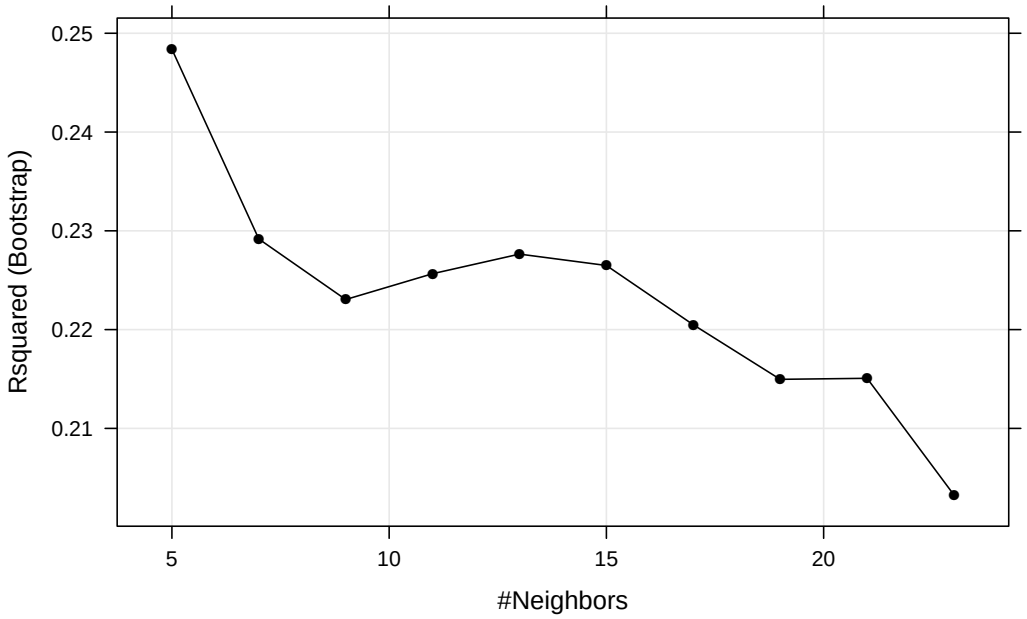


Figure 14: K-nearest neighbors tuning parameter profile for the permeability data

Exercise 5

Exercise 6.3 describes data for a chemical manufacturing process. Use the same data imputation, data–splitting and pre–processing steps as before and train several non–linear regression models.

(a) Which non–linear regression model gives the optimal resampling and test set performance?

(b) Which predictors are most important in the optimal non–linear regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear model?

(c) Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal non–linear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

Solutions

```
> library(AppliedPredictiveModeling)
> data(ChemicalManufacturingProcess)
>
> predictors <- subset(ChemicalManufacturingProcess,select= -Yield)
> yield <- subset(ChemicalManufacturingProcess,select="Yield")
>
> set.seed(517)
> trainingRows <- createDataPartition(yield$Yield,
+                                     p = 0.7,
+                                     list = FALSE)
>
> trainPredictors <- predictors[trainingRows,]
> trainYield <- yield[trainingRows,]
>
> testPredictors <- predictors[-trainingRows,]
> testYield <- yield[-trainingRows,]
>
> #Pre-process trainPredictors and apply to trainPredictors and testPredictors
> pp <- preProcess(trainPredictors,method=c("BoxCox","center","scale","knnImpute"))
> ppTrainPredictors <- predict(pp,trainPredictors)
> ppTestPredictors <- predict(pp,testPredictors)
>
> #Identify and remove NZV
> nzvpp = nearZeroVar(ppTrainPredictors)
> ppTrainPredictors <- ppTrainPredictors[-nzvpp]
> ppTestPredictors <- ppTestPredictors[-nzvpp]
>
> #Identify and remove highly correlated predictors
> predcorr = cor(ppTrainPredictors)
```

```
> highCorrpp <- findCorrelation(predcorr)
> ppTrainPredictors <- ppTrainPredictors[, -highCorrpp]
> ppTestPredictors <- ppTestPredictors[, -highCorrpp]
>
> #Set-up trainControl
> set.seed(517)
> ctrl <- trainControl(method = "boot", number = 25)
```

For this example, we will tune a MARS and SVM (polynomial kernel function) model

```
> set.seed(614)
>
> marsChemGrid <- expand.grid(degree = c(1:2), nprune = c(2:10))
> marsChemTune <- train(x = ppTrainPredictors, y = trainYield,
+                       method = "earth",
+                       tuneGrid = marsChemGrid,
+                       trControl = ctrl)
>
> psvmTuneGrid <- expand.grid(C=c(0.01,0.05,0.1), degree=c(1,2), scale=c(0.25,0.5,1))
> PSVMChemTune <- train(x = ppTrainPredictors, y = trainYield,
+                       method = "svmPoly",
+                       trControl = ctrl,
+                       tuneGrid = psvmTuneGrid)
```

Figure 15 indicates that the optimal degree and number of terms that maximize R^2 are 1 and 4, respectively, with an R^2 of 0.51. Hence, MARS identifies a fairly simple model with the terms listed in Table 1.

The test set predictions for the MARS model are presented in Figure 16, with an R^2 value of 0.583.

	y
(Intercept)	38.07
h(ManufacturingProcess32–0.802862)	1.34
h(ManufacturingProcess09–1.03728)	0.82
h(ManufacturingProcess17-1.00337)	-1.21

Table 1: MARS coefficients for chemical manufacturing data.

The final MARS model has slightly worse cross-validation and test set performance than the optimal PLS model (see solutions for Exercise 6.3). This would indicate that the underlying structure between the predictors and the response is approximately linear. We can use marginal plots of each predictor in the model to better understand the relationship that the MARS model has detected. Figure 17 displays the marginal relationships between the predictors identified by the model and the predicted response. Based on this figure, the underlying relationships are approximately linear in this model.

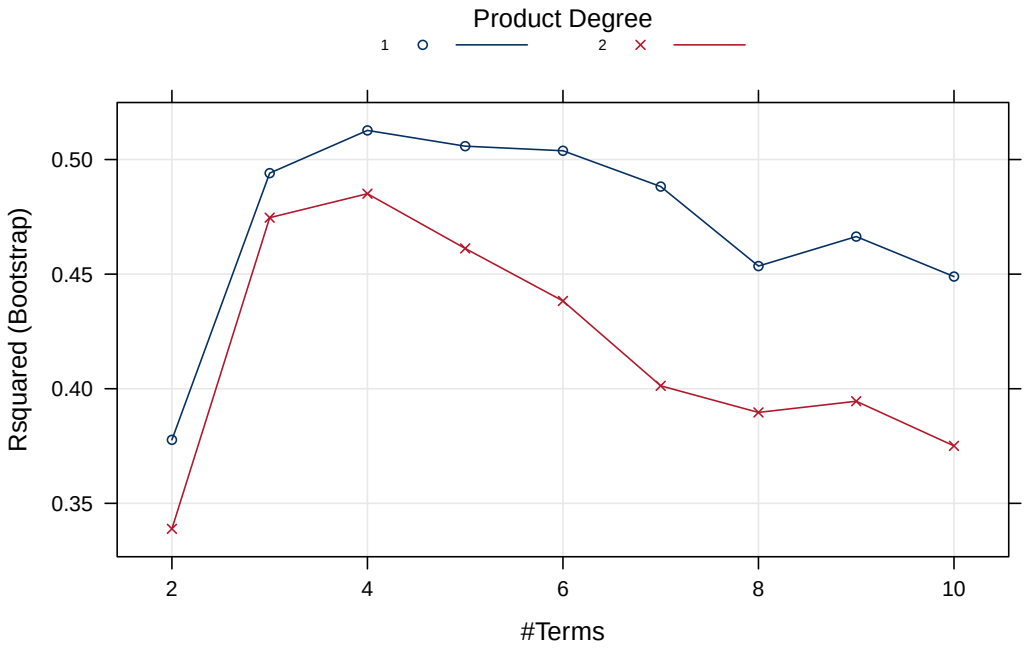


Figure 15: MARS tuning parameter profile for the chemical manufacturing data.

Referring back to Figure 15 of the solutions from Chapter 6, the top two PLS predictors are ManufacturingProcess32 and ManufacturingProcess09, which are the same as what the MARS model identifies. PLS, however, identifies additional predictive information from the other predictors that improve the predictive ability of the models. Overall, many of the manufacturing process predictors are at the top of the importance list.

Figure 18 indicates that the optimal degree, cost, and scale that maximize R^2 are 1, 0.01, and 0.5, respectively, with an R^2 of 0.44.

The polynomial SVM tuning parameters both indicate that a linear model is sufficient for this model. Specifically, a degree of 1 is an indicator of linear structure, and a low cost is also an indicator of linear structure.

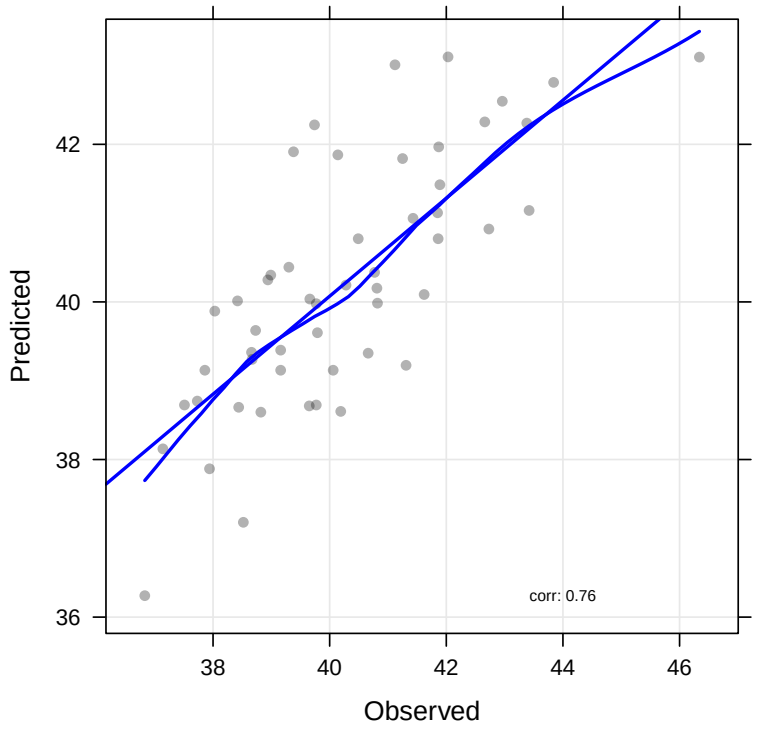


Figure 16: MARS predictions for the test set for the chemical manufacturing data.

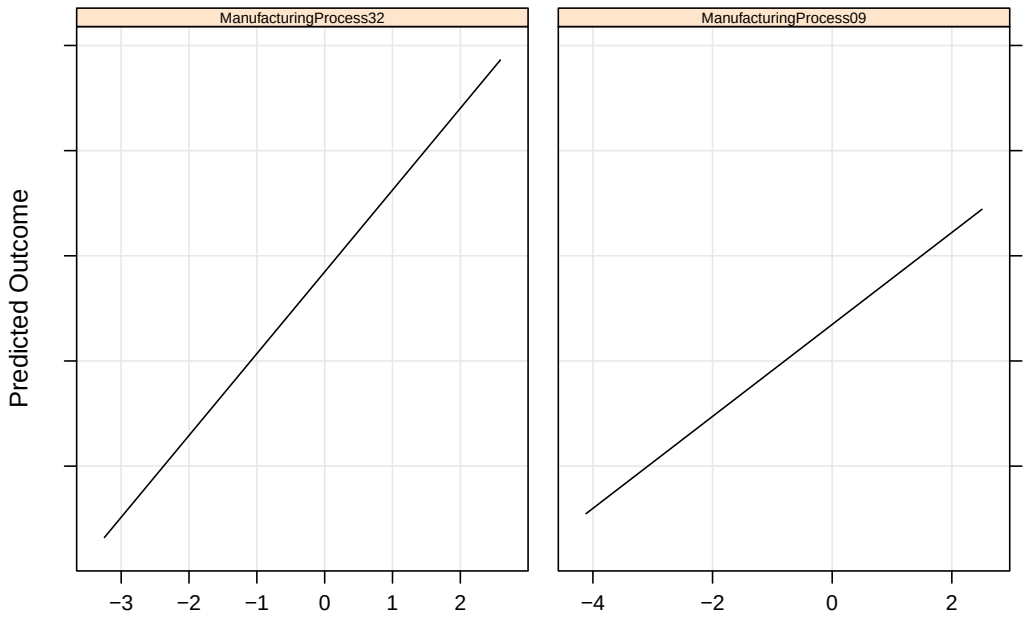


Figure 17: MARS marginal plots for chemical manufacturing data.

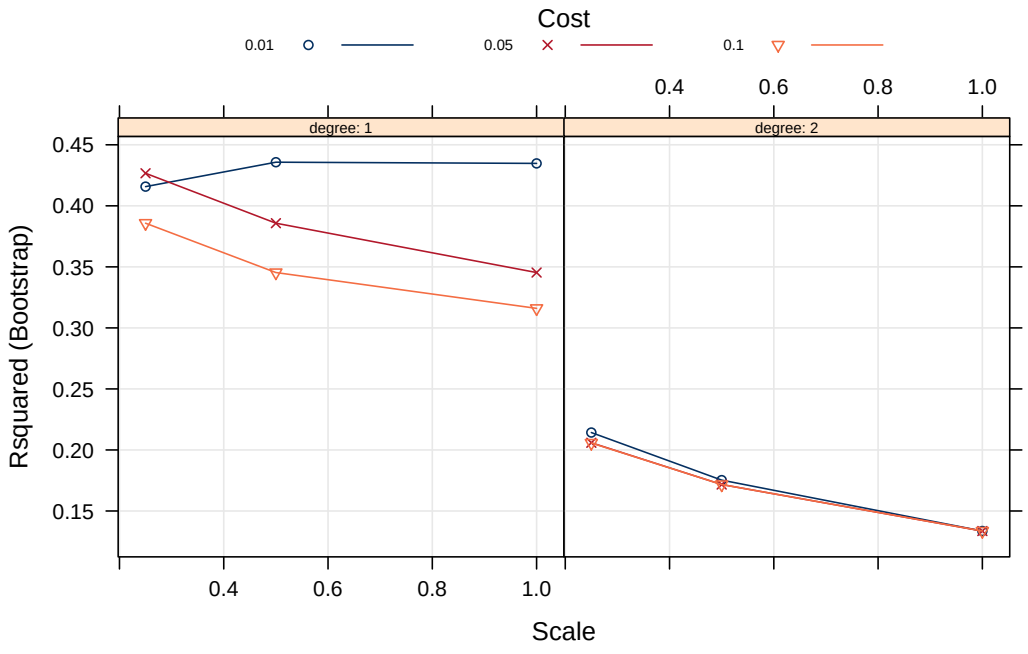


Figure 18: Polynomial SVM tuning parameter profile for the chemical manufacturing data.

Session Info

- R Under development (unstable) (2014-12-29 r67265), `x86_64-apple-darwin10.8.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: AppliedPredictiveModeling 1.1-6, caret 6.0-41, corrplot 0.73, doMC 1.3.3, earth 4.1.0, elasticnet 1.1, foreach 1.4.2, ggplot2 1.0.0, iterators 1.0.7, kernlab 0.9-19, knitr 1.6, lars 1.2, lattice 0.20-29, latticeExtra 0.6-26, mlbench 2.1-1, nnet 7.3-8, plotmo 2.2.0, plotrix 3.5-7, pls 2.4-3, RColorBrewer 1.0-5, reshape2 1.4, xtable 1.7-3
- Loaded via a namespace (and not attached): BradleyTerry2 1.0-5, brglm 0.5-9, car 2.0-21, class 7.3-11, cluster 1.15.3, codetools 0.2-9, colorspace 1.2-4, compiler 3.2.0, CORElearn 0.9.43, digest 0.6.4, e1071 1.6-3, evaluate 0.5.5, formatR 0.10, grid 3.2.0, gtable 0.1.2, gtools 3.4.1, highr 0.3, lme4 1.1-7, MASS 7.3-35, Matrix 1.1-4, minqa 1.2.3, munsell 0.4.2, nlme 3.1-118, nloptr 1.0.4, plyr 1.8.1, proto 0.3-10, RANN 2.4.1, Rcpp 0.11.2, rpart 4.1-8, scales 0.2.4, splines 3.2.0, stringr 0.6.2, tools 3.2.0