KJ_Chapter7_Problem2

Jonathan Hernandez

7.2. Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data:

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

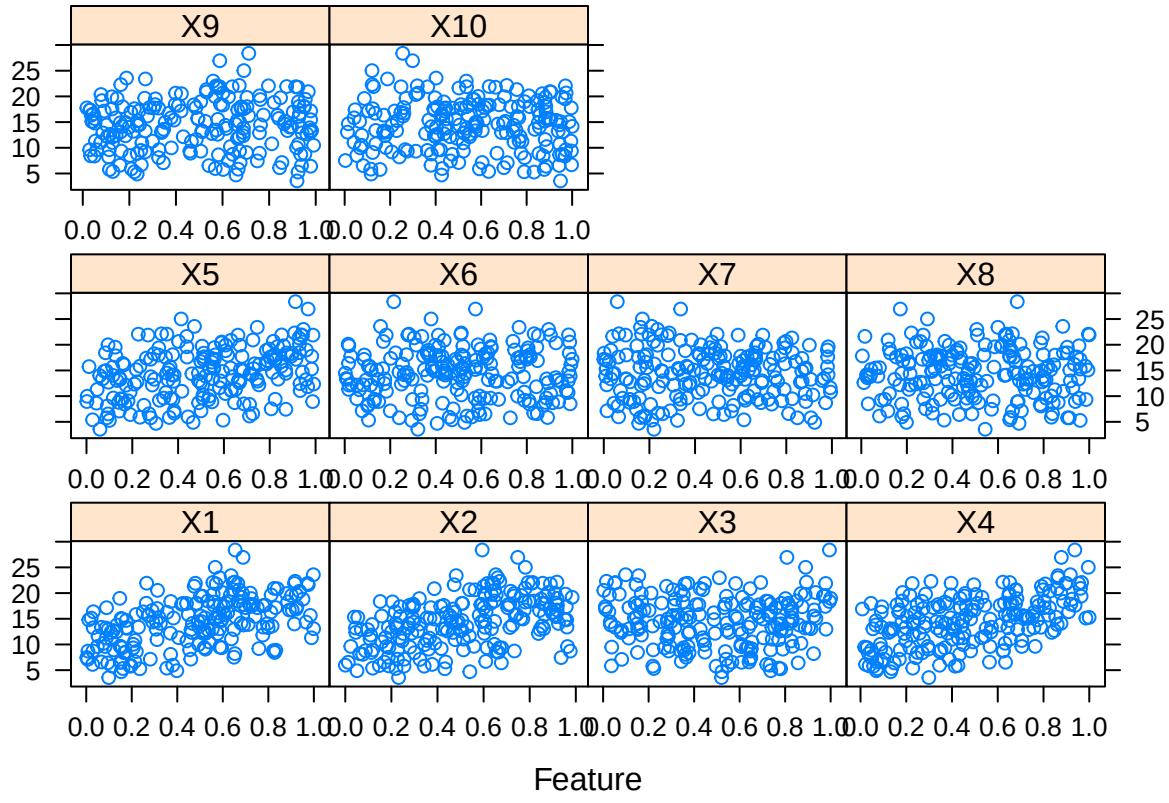
where the x values are random variables uniformly distributed between [0,1] (there are also 5 other non-informative variables also created in the simulation). The package mlbench contains a function called mlbench.friedman1 that simulates these data:

```
library(mlbench)
library(caret)
```

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

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

```
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:

```
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 (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  3.565620  0.4887976  2.886629
```

```
##    7  3.422420  0.5300524  2.752964
##    9  3.368072  0.5536927  2.715310
##   11  3.323010  0.5779056  2.669375
##   13  3.275835  0.6030846  2.628663
##   15  3.261864  0.6163510  2.621192
##   17  3.261973  0.6267032  2.616956
##   19  3.286299  0.6281075  2.640585
##   21  3.280950  0.6390386  2.643807
##   23  3.292397  0.6440392  2.656080
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 15.
```

```
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)
```

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

- The approach I will use is to look at different modeling methods for nonlinear regression such as Neural Networks, SVM with different kernels and MARS (Multivariate Adaptive Regression Splines) and make predictions as well as compute the RMSE and R^2 values.
- In using a Neural Network, we will use the train() function with the method = "nnet" to specify we are training a model as a neural network. The max number of parameters to estimate is based on $H(P+1)+H+1$ where H is the number of hidden layers we want and, P is the number of predictors.

```
set.seed(200)
# Train a neural network model where I specify 10 hidden layers and all predictors
# inputted to each hiden layer (linear combinations of predictors)
neuralnet_model <- train(x=trainingData$x, y=trainingData$y, method = "nnet",
                         preProc = c("center", "scale"), linout=TRUE,
                         MaxNWts = 10*(ncol(trainingData$x)+1) + 10 + 1,
                         maxit=500, trace=FALSE)
neuralnet_model
```

```
## Neural Network
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE      Rsquared   MAE
##   1     0e+00  2.632680  0.7143900  2.058601
##   1     1e-04  2.728016  0.6963892  2.128850
##   1     1e-01  2.500573  0.7462985  1.929959
##   3     0e+00  2.882422  0.6845956  2.269226
```

```
##   3     1e-04  2.824605  0.6920881  2.218588
##   3     1e-01  2.818867  0.6915981  2.195091
##   5     0e+00  5.396666  0.5166371  3.227736
##   5     1e-04  4.203312  0.5298866  2.989422
##   5     1e-01  3.039955  0.6685665  2.377394
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 1 and decay = 0.1.
```

```
# predict with our neural network model
neuralnet_pred <- predict(neuralnet_model, newdata=testData$x)
postResample(pred = neuralnet_pred, obs = testData$y)
```

```
##      RMSE  Rsquared       MAE
## 2.6492918 0.7177264 2.0295117
```

- Looking at the neural network model, after scaling/centering, and setting the max number of hidden layers to be 10, we see that train() picked the model with one hidden layer and a decay value of 0.1 that minimize the RMSE.
- Now, let's look at a SVM model that uses the kernel function as the radial basis function by using 3 kernels, Radial, polynomial and linear:

```
library(kernlab)
```

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

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

```
set.seed(200)
# Train SVM models using different kernels
# Note: you may need to install the kernlab package in order to use
# method = svmRadial, svmPoly and svmLinear
svm_model_radial <- train(x=trainingData$x, y=trainingData$y, method = "svmRadial",
                          preProc = c("center", "scale"), tunelength=10)

svm_model_poly <- train(x=trainingData$x, y=trainingData$y, method = "svmPoly",
                          preProc = c("center", "scale"), tunelength=10)

svm_model_linear <- train(x=trainingData$x, y=trainingData$y, method = "svmLinear",
                          preProc = c("center", "scale"), tunelength=10)

svm_model_radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
```

```
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  2.580633  0.7683393  2.009726
##   0.50  2.377472  0.7816465  1.842621
##   1.00  2.245113  0.7991788  1.740655
##
## Tuning parameter 'sigma' was held constant at a value of 0.06574662
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.06574662 and C = 1.
```

```
svm_model_poly
```

```
## Support Vector Machines with Polynomial Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   degree  scale  C     RMSE      Rsquared   MAE
##   1       0.001  0.25  4.792128  0.7168723  3.950158
##   1       0.001  0.50  4.622801  0.7201153  3.808598
##   1       0.001  1.00  4.313849  0.7254661  3.544288
##   1       0.010  0.25  3.606573  0.7326320  2.951303
##   1       0.010  0.50  2.968362  0.7403258  2.435437
##   1       0.010  1.00  2.630987  0.7470647  2.142283
##   1       0.100  0.25  2.549468  0.7456426  2.056187
##   1       0.100  0.50  2.544599  0.7465421  2.043769
##   1       0.100  1.00  2.544604  0.7477025  2.037695
##   2       0.001  0.25  4.622802  0.7201204  3.808608
##   2       0.001  0.50  4.313824  0.7254854  3.544282
##   2       0.001  1.00  3.798887  0.7329190  3.109682
##   2       0.010  0.25  2.966687  0.7419532  2.433549
##   2       0.010  0.50  2.617770  0.7503528  2.131274
##   2       0.010  1.00  2.525091  0.7519388  2.037442
##   2       0.100  0.25  2.248812  0.7965482  1.748205
##   2       0.100  0.50  2.233501  0.8001700  1.730344
##   2       0.100  1.00  2.236572  0.8022526  1.747894
##   3       0.001  0.25  4.462960  0.7245693  3.672234
##   3       0.001  0.50  4.037269  0.7294667  3.309809
##   3       0.001  1.00  3.438127  0.7357299  2.817646
##   3       0.010  0.25  2.701373  0.7504215  2.204180
##   3       0.010  0.50  2.526618  0.7558180  2.042530
##   3       0.010  1.00  2.443114  0.7648298  1.956653
##   3       0.100  0.25  2.144078  0.8147117  1.665053
##   3       0.100  0.50  2.160520  0.8137549  1.673245
##   3       0.100  1.00  2.209038  0.8071517  1.717293
```

```
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were degree = 3, scale = 0.1 and C
##  = 0.25.
```

```
svm_model_linear
```

```
## Support Vector Machines with Linear Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   2.529319  0.7540623  2.030249
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
# predict with our SVM models
svm_pred_radial <- predict(svm_model_radial, newx=testData$x)
svm_pred_poly <- predict(svm_model_poly, newx=testData$x)
svm_pred_linear <- predict(svm_model_linear, newx=testData$x)
```

- We see that by using SVM, using kernels such as ploynomial and radial show to have low RMSE and high R^2 values compared to using a linear kernel. This can probably be due to the fact that the data is nonlinear.
- Finally let's see a MARS model in action (Multivariate Adaptive Regression Splines) and see how it compares to the other models in regards to RMSE and R^2

```
# Fit a MARS model
library(earth)
```

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

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

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

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

```
mars_model <- train(x=trainingData$x, y=trainingData$y, method="earth",
                    preProc=c("center", "scale"))
mars_model
```

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   nprune  RMSE      Rsquared   MAE
##    2      4.422592  0.2127902  3.636756
##    8      1.864401  0.8570752  1.443484
##   15      1.789808  0.8709314  1.392745
##
## Tuning parameter 'degree' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 15 and degree = 1.
```

```
# compute RMSE and R^2 for MARS model
mars_pred <- predict(mars_model, newdata=testData$x)
postResample(pred=mars_pred, obs=testData$y)
```

```
##      RMSE  Rsquared       MAE
## 1.8136467 0.8677298 1.3911836
```

- Using the MARS model, let's see which coefficients were important as well as the hinge functions used and coefficients.

```
mars_summary <- earth(trainingData$x, trainingData$y)
#summary(mars_summary) # check out hinge functions used and coefficients
varImp(mars_model)
```

```
## earth variable importance
##
##     Overall
## X1   100.00
## X4    84.22
## X2    67.22
## X5    45.44
## X3    34.63
## X6    11.90
## X8     0.00
## X10    0.00
## X7     0.00
## X9     0.00
```

- We see that MARS did select X1-X5 as important variables/predictors from using the varImp() function.
- Final summary of each model and RMSE and R^2 based on predictions using the test data.

```r
library(kableExtra)
# use postResampe() and append each result to be a dataframe
model_metrics_summary <-
  rbind(postResample(pred=knnPred, obs=testData$y),
        postResample(pred=neuralnet_pred, obs=testData$y),
        postResample(pred=svm_pred_radial, obs=testData$y),
        postResample(pred=svm_pred_poly, obs=testData$y),
        postResample(pred=svm_pred_linear, obs=testData$y),
        postResample(pred=mars_pred, obs=testData$y))

# convert to data frame
model_metrics_summary <- data.frame(model_metrics_summary)

# put in name of each model and then sort them based on RMSE
rownames(model_metrics_summary) <- c("k-NN", "Neural Network", "SVM-Radial",
                                     "SVM-Polynomial", "SVM-Linear", "MARS")

model_metrics_summary[order(-model_metrics_summary$RMSE), ] %>% kable() %>%
  kable_styling(fixed_thead = T)
```

	RMSE	Rsquared	MAE
SVM-Polynomial	6.770847	NA	5.446810
SVM-Linear	6.643233	NA	5.347818
SVM-Radial	6.568821	NA	5.290032
k-NN	3.175066	0.6785946	2.544317
Neural Network	2.649292	0.7177264	2.029512
MARS	1.813647	0.8677298	1.391184

- The MARS model has the best performance in regards to RMSE and R^2 values. SVM using polynomial, radial, neural network, SVM using linear and k-NN are the 2nd, 3rd, 4th, 5th and 6th best performing model respectively.