KJ 7.2

Li exl i vKi mki v

Libraries

Load libraries.

```
library(mlbench)
library(caret)
library(earth)
library(ggplot2)
library(tidyr)
library(dplyr)
```

Set up question data.

Set seed.

```
set.seed(200)
```

Set up training data.

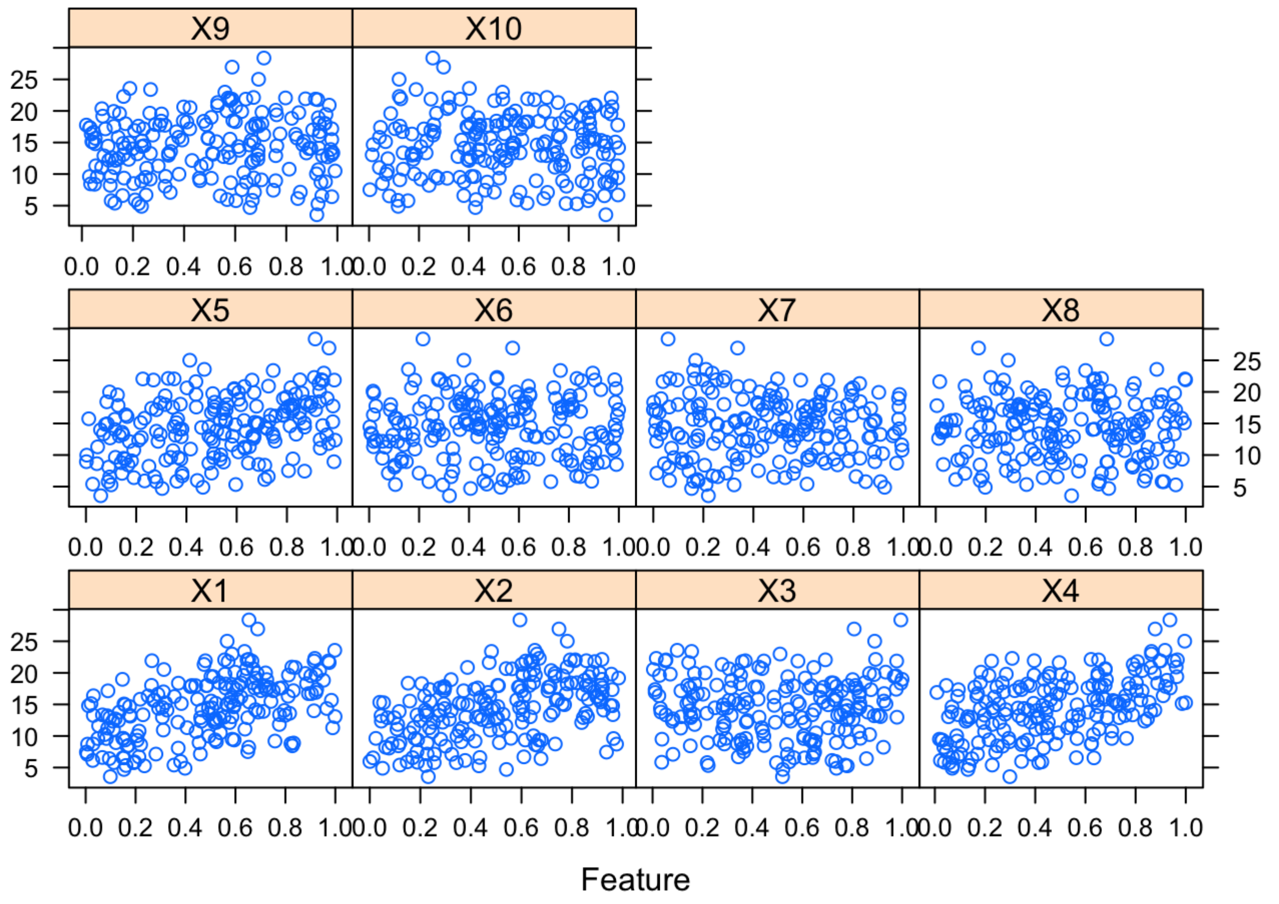
```
trainingData <- mlbench.friedman1(200, sd = 1)
```

Format training data.

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

Plot training data predictors vs. outcome.

```
featurePlot(trainingData$x,trainingData$y)
```



Looks like the first five predictors have a better correlation with the outcome than the last five.

Set up and format test data.

```
testData <- mlbench.friedman1(5000, sd = 1)
testData$x <- data.frame(testData$x)
```

Instructions

Tune several models on these data.

Example of a model given was using the caret package train function with a KNN model, pre-process training data with centering and scaling, tuneLength = 10 (for KNN, this would mean trying 10 different values for k).

Then, use the models based on the training data to predict on test data.

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

Answer

Preliminary steps

Before we start creating models, let's set our own seed, separate from the one used to create the input data.

```
set.seed(1392)
```

Also, write a function to look at error and R-squared of a given model applied to the test data.

```
error_and_Rsquared <- function(model){
    predictions <- predict(model,newdata = testData$x)
    return(postResample(pred = predictions, obs = testData$y))
}
```

Neural networks

Let's try the neural networks method with model averaging.

```
nnetGrid <- expand.grid(.decay = c(0, 0.01, .1),
                .size = c(1:10),
                .bag = FALSE)

averaging_nnet_model <- train(trainingData$x,trainingData$y,
                            method="avNNet",
                            tuneGrid = nnetGrid,
                            trControl = trainControl(method = "cv", number = 10),
                            preProc = c("center", "scale"),
                            linout = TRUE,
                            trace=FALSE,
                            maxit = 500)
```

Multivariate Adaptive Regression Splines (MARS)

Use the earth command from the earth package.

```
MARS_model <- earth(trainingData$x,trainingData$y)
```

Print model details.

```
MARS_model
```

```
## Selected 12 of 18 terms, and 6 of 10 predictors
## Termination condition: Reached nk 21
## Importance: X1, X4, X2, X5, X3, X6, X7-unused, X8-unused, X9-unused, ...
## Number of terms at each degree of interaction: 1 11 (additive model)
## GCV 2.540556    RSS 397.9654    GRSq 0.8968524    RSq 0.9183982
```

```
summary(MARS_model)
```

```
## Call: earth(x=trainingData$x, y=trainingData$y)
##
##                coefficients
## (Intercept)       18.451984
## h(0.621722-X1)   -11.074396
## h(0.601063-X2)   -10.744225
## h(X3-0.281766)    20.607853
## h(0.447442-X3)    17.880232
## h(X3-0.447442)   -23.282007
## h(X3-0.636458)    15.150350
## h(0.734892-X4)   -10.027487
## h(X4-0.734892)     9.092045
## h(0.850094-X5)    -4.723407
## h(X5-0.850094)    10.832932
## h(X6-0.361791)    -1.956821
##
## Selected 12 of 18 terms, and 6 of 10 predictors
## Termination condition: Reached nk 21
## Importance: X1, X4, X2, X5, X3, X6, X7-unused, X8-unused, X9-unused, ...
## Number of terms at each degree of interaction: 1 11 (additive model)
## GCV 2.540556    RSS 397.9654    GRSq 0.8968524    RSq 0.9183982
```

"Does MARS select the informative predictors (those named X1–X5)?"

For the most part, yes, but not entirely. The MARS model does include all five of the informative predictors (X1-X5) and excludes most of the last predictors that did not appear as correlated with outcome. However, it does also include one of the last predictors (X6).

Support Vector Machine (SVM)

I tried running SVM with a polynomial kernel here, but runtime was impractically long.

However, I will try running with both a linear and radial basis function kernel.

```
SVM_linear_model <- train(trainingData$x,trainingData$y,
                          method="svmLinear",
                          trControl = trainControl(method = "cv", number = 10),
                          preProc = c("center", "scale"),
                          tuneLength = 14)
```

```
SVM_RBF_model <- train(trainingData$x,trainingData$y,
                       method="svmRadial",
                       trControl = trainControl(method = "cv", number = 10),
                       preProc = c("center", "scale"),
                       tuneLength = 14)
```

K-Nearest Neighbors (KNN)

Let's run KNN just as it was run in the question example.

```
knn_model <- train(x = trainingData$x,
                y = trainingData$y,
                method = "knn",
                preProc = c("center", "scale"),
                tuneLength = 10)
```

Comparing models

Run function to get error and R-squared for each model.

```
for(model in c("averaging_nnet_model","MARS_model","SVM_linear_model","SVM_RBF_model","k
nn_model"))
{
    print(model)
    print(error_and_Rsquared(get(model)))
}
```

```
## [1] "averaging_nnet_model"
##     RMSE Rsquared      MAE
## 2.110355 0.824974 1.581060
## [1] "MARS_model"
##      RMSE  Rsquared       MAE
## 1.8136467 0.8677298 1.3911836
## [1] "SVM_linear_model"
##      RMSE  Rsquared       MAE
## 2.7633860 0.6973384 2.0970616
## [1] "SVM_RBF_model"
##     RMSE Rsquared      MAE
## 2.092662 0.822730 1.590366
## [1] "knn_model"
##      RMSE  Rsquared       MAE
## 3.2040595 0.6819919 2.5683461
```

Looks like the best model here in terms of both R-squared (aka variance explained) and RMSE/MAE (minimizing error) within the test data is MARS.

After that, the neural network with model averaging and SVM with radial basis function both perform pretty similarly, falling slightly behind MARS but still doing pretty well.

The KNN and SVM with linear kernel models perform the worst out of the five models tested.