HW2

Christopher Bloome

7/8/2021

# 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 = 10 sin(πx1x2) + 20(x3 − 0.5)2 + 10x4 + 5x5 + N(0, σ2) 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)

## Warning: package 'mlbench' was built under R version 4.0.5

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)

## Loading required package: lattice

## Loading required package: ggplot2

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 3.466085 0.5121775 2.816838  
## 7 3.349428 0.5452823 2.727410  
## 9 3.264276 0.5785990 2.660026  
## 11 3.214216 0.6024244 2.603767  
## 13 3.196510 0.6176570 2.591935  
## 15 3.184173 0.6305506 2.577482  
## 17 3.183130 0.6425367 2.567787  
## 19 3.198752 0.6483184 2.592683  
## 21 3.188993 0.6611428 2.588787  
## 23 3.200458 0.6638353 2.604529  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 17.

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 MAE   
## 3.2040595 0.6819919 2.5683461

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

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

## 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 3.596096 0.4930100 2.930570  
## 7 3.459955 0.5319522 2.798379  
## 9 3.368640 0.5669301 2.726710  
## 11 3.357390 0.5808369 2.723971  
## 13 3.355299 0.5920307 2.726627  
## 15 3.343154 0.6053457 2.730079  
## 17 3.313771 0.6249174 2.703343  
## 19 3.305186 0.6393157 2.701323  
## 21 3.306140 0.6497242 2.709582  
## 23 3.328424 0.6524463 2.730748  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 19.

svmR <- train(x=trainingData$x, y=trainingData$y,   
 method="svmRadial",   
 preProcess=c("center", "scale"),   
 tuneLength=20)  
svmR

## 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 2.618175 0.7723520 2.077132  
## 0.50 2.401710 0.7873751 1.896034  
## 1.00 2.259275 0.8052616 1.783054  
## 2.00 2.184530 0.8149557 1.716331  
## 4.00 2.153590 0.8196078 1.693556  
## 8.00 2.147415 0.8205385 1.690739  
## 16.00 2.146507 0.8206974 1.690291  
## 32.00 2.146507 0.8206974 1.690291  
## 64.00 2.146507 0.8206974 1.690291  
## 128.00 2.146507 0.8206974 1.690291  
## 256.00 2.146507 0.8206974 1.690291  
## 512.00 2.146507 0.8206974 1.690291  
## 1024.00 2.146507 0.8206974 1.690291  
## 2048.00 2.146507 0.8206974 1.690291  
## 4096.00 2.146507 0.8206974 1.690291  
## 8192.00 2.146507 0.8206974 1.690291  
## 16384.00 2.146507 0.8206974 1.690291  
## 32768.00 2.146507 0.8206974 1.690291  
## 65536.00 2.146507 0.8206974 1.690291  
## 131072.00 2.146507 0.8206974 1.690291  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.07301549  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were sigma = 0.07301549 and C = 16.

mars\_model <- train(x=trainingData$x,   
 y=trainingData$y,   
 method="earth",  
 preProcess = c("center", "scale"))

## Loading required package: earth

## Warning: package 'earth' was built under R version 4.0.5

## Loading required package: Formula

## Loading required package: plotmo

## Warning: package 'plotmo' was built under R version 4.0.5

## Loading required package: plotrix

## Loading required package: TeachingDemos

## Warning: package 'TeachingDemos' was built under R version 4.0.5

mars\_predict <- predict(mars\_model, newdata=testData$x)  
mars\_results <- postResample(pred=mars\_predict, obs=testData$y)  
mars\_results

## RMSE Rsquared MAE   
## 1.8136467 0.8677298 1.3911836

The RMSE Was the best in the MARS model.

varImp(mars\_model, 10)

## earth variable importance  
##   
## Overall  
## X1 100.00  
## X4 82.08  
## X2 62.79  
## X5 38.07  
## X3 25.80  
## X6 0.00

Within the MARS model, ONLY the X1-X5 were important, each of the others were effectively discarded.

# 7.2 Python

We take the same dataset in 7.2 and try to reproduce the questions using Python. To do this we used the reticulate package in R that enables the use of using python code in markdown chunks

Overall the results were lackluster due to the differences between how R and the Sklearn packages handle variables. The sklearn package expects target variables not to be continuous. The target variables are encoded to integer values and thus the package treats each target variable as a unique class. Since there are 200 observations in the training dataset, the encoding process created 200 different classes for the target variables. Additionally due to this, hyperparameter tuning was not possible. This overall resulted in very poor training performance for both SVM and KNN models, the MARS model faired the best

library(reticulate)

## Warning: package 'reticulate' was built under R version 4.0.5

use\_python("C:/Users/dhair/anaconda3") #should be updated per your lcoal env  
  
#Once installed can be removed  
py\_install("pandas")  
py\_install("scikit-learn")  
py\_install("sklearn-contrib-py-earth")

Transfer training and test dataframes from the R enviornment to the Python enviornemnt

py$train\_set\_x <- r\_to\_py(trainingData$x)

## Warning: Python 'C:/Users/dhair/anaconda3/python.exe' was requested but 'C:/  
## Users/dhair/AppData/Local/r-miniconda/envs/r-reticulate/python.exe' was loaded  
## instead (see reticulate::py\_config() for more information)

py$train\_set\_y <- r\_to\_py(trainingData$y)  
py$test\_set <- r\_to\_py(testData$x)

Import python libraries

import pandas as pd  
import numpy as np  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
import pyearth  
from sklearn.model\_selection import RandomizedSearchCV  
from sklearn.model\_selection import GridSearchCV  
from sklearn import preprocessing  
from sklearn import utils  
from sklearn.metrics import mean\_squared\_error  
from sklearn.metrics import r2\_score  
from sklearn.metrics import classification\_report  
from sklearn import preprocessing

Scale training data and encode the target variable

scaler = preprocessing.StandardScaler().fit(train\_set\_x)  
train\_set\_x\_scaled = scaler.transform(train\_set\_x)  
  
encoder = preprocessing.LabelEncoder()  
train\_set\_y\_encoded = encoder.fit\_transform(train\_set\_y)

KNN model results in scores that do not make sense, due to the fact that there are no similar target classes becuase of the encoding

knn = KNeighborsClassifier(n\_neighbors=19)  
knn\_results = knn.fit(train\_set\_x\_scaled, train\_set\_y\_encoded)  
knn\_pred = knn\_results.predict(train\_set\_x\_scaled)  
print("KNN Score: " + str(knn.score(train\_set\_x\_scaled, train\_set\_y\_encoded)))

## KNN Score: 0.015

print("KNN RMSE: " + str(mean\_squared\_error(train\_set\_y\_encoded, knn\_pred)))

## KNN RMSE: 8153.47

print("KNN R-squared: " + str(r2\_score(train\_set\_y\_encoded, knn\_pred)))

## KNN R-squared: -1.4461021525538138

THe SVM model results in the same issues as the KNN model

svm\_m = SVC(kernel='rbf', gamma="auto")  
svm\_results = svm\_m.fit(train\_set\_x\_scaled, train\_set\_y\_encoded)  
  
print("SVM Score: " + str(svm\_results.score(train\_set\_x\_scaled, train\_set\_y\_encoded)))

## SVM Score: 1.0

svm\_pred = svm\_results.predict(train\_set\_x\_scaled)  
print("SVM RMSE: " + str(mean\_squared\_error(train\_set\_y\_encoded, svm\_pred)))

## SVM RMSE: 0.0

print("SVM R-squared: " + str(r2\_score(train\_set\_y\_encoded, svm\_pred)))

## SVM R-squared: 1.0

The MARS model fairs the best of the 3 given the dataset

mars = pyearth.Earth()  
mars\_results = mars.fit(train\_set\_x\_scaled, train\_set\_y)

## C:\Users\dhair\AppData\Local\R-MINI~1\envs\R-RETI~1\lib\site-packages\pyearth\earth.py:813: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.  
## To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.  
## pruning\_passer.run()  
## C:\Users\dhair\AppData\Local\R-MINI~1\envs\R-RETI~1\lib\site-packages\pyearth\earth.py:1066: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.  
## To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.  
## coef, resid = np.linalg.lstsq(B, weighted\_y[:, i])[0:2]

print(mars\_results.summary())

## Earth Model  
## --------------------------------------  
## Basis Function Pruned Coefficient   
## --------------------------------------  
## (Intercept) No 12.4128   
## h(x0-0.673905) Yes None   
## h(0.673905-x0) No -2.01315   
## x3 No 2.76798   
## h(x1-0.342446) Yes None   
## h(0.342446-x1) No -2.88561   
## x4 Yes None   
## h(x2+0.217285) No 3.5214   
## h(-0.217285-x2) No -5.43924   
## h(x8-1.39745) Yes None   
## h(1.39745-x8) Yes None   
## h(x0+0.462884) Yes None   
## h(-0.462884-x0) No -1.86342   
## h(x2+0.806189) Yes None   
## h(-0.806189-x2) No 5.89439   
## h(x2-0.421124) Yes None   
## h(0.421124-x2) No 4.63714   
## x5 Yes None   
## h(x5+0.452668) Yes None   
## h(-0.452668-x5) No -1.1895   
## h(x5-1.61069) Yes None   
## h(1.61069-x5) No 0.700106   
## h(x5-1.43655) Yes None   
## h(1.43655-x5) Yes None   
## h(x4-0.537899) Yes None   
## h(0.537899-x4) Yes None   
## h(x4+0.720508) No 1.4635   
## h(-0.720508-x4) No -1.3749   
## --------------------------------------  
## MSE: 1.9458, GCV: 2.7251, RSQ: 0.9202, GRSQ: 0.8894