DATA624: Homework 8

Donald Butler

2023-11-12

Contents

Homework 8

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(AppliedPredictiveModeling)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(mlbench)
library(earth)
```

```
## Loading required package: Formula
## Loading required package: plotmo
## Loading required package: plotrix
## Loading required package: TeachingDemos
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
##
## The following object is masked from 'package:purrr':
##
##     cross
##
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

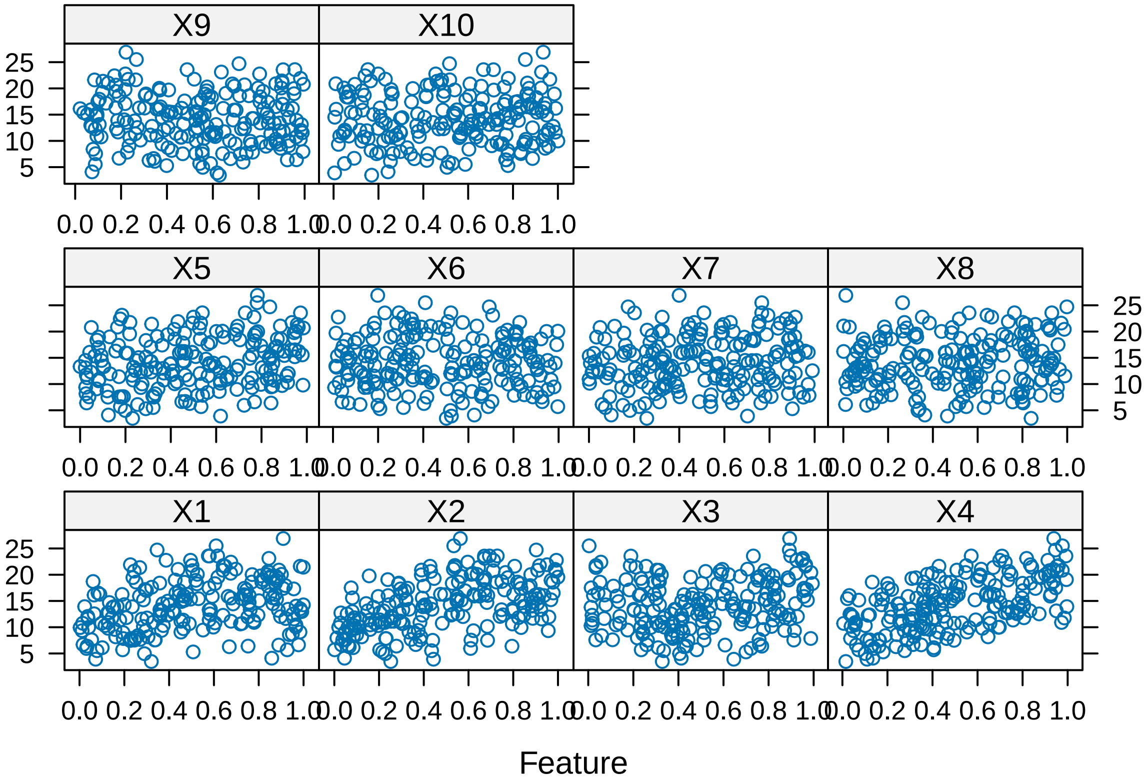
Exercise 7.2

Friedman (1991) introduced several benchmark data sets created 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:

```
set.seed(31415)
trainingData <- mlbench.friedman1(200, sd = 1)
## We convert the 'x' data from a matrix to a data frame
## One reason is taht this will give the column 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:

KNN: k-Nearest Neighbors

```
knnModel <- train(x = trainingData$x,
                  y = trainingData$y,
                  method = 'knn',
                  preProcess = 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  3.719008  0.4517539  2.966565
##    7  3.619962  0.4804642  2.893141
##    9  3.553036  0.5054952  2.853502
##   11  3.532333  0.5216344  2.841214
##   13  3.510321  0.5378856  2.824333
##   15  3.493860  0.5530622  2.815556
##   17  3.481650  0.5688741  2.825945
##   19  3.489310  0.5757182  2.837525
##   21  3.497301  0.5808056  2.856954
##   23  3.492655  0.5925379  2.861990
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 17.
```

```r
knnPred <- predict(knnModel, newdata = testData$x)
## The function 'postResample can be used to get the test set
## performance values
knnResult <- data.frame(as.list(postResample(pred = knnPred, obs = testData$y))) |>
  mutate(model = 'knn') |>
  relocate(model, RMSE, Rsquared, MAE)
knnResult
```

```
##   model     RMSE  Rsquared      MAE
## 1   knn 3.241683 0.6701753 2.600879
```

NNET: Neural Networks

```r
nnetTune <- train(trainingData$x, trainingData$y,
                  method = 'nnet',
                  tuneGrid = expand.grid(.decay = c(0, 0.01, .1), .size = c(1:10)),
                  trControl = trainControl(method = 'cv', number = 10),
                  preProcess = c('center', 'scale'),
                  linout = TRUE, trace = FALSE,
                  MaxNWts = 10 * (ncol(trainingData$x) + 1) + 10 + 1,
                  maxit = 500)
nnetTune
```

```
## Neural Network
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
```

```
##
##   decay  size  RMSE      Rsquared   MAE
##   0.00    1    2.691690  0.7155733  2.147649
##   0.00    2    2.960342  0.6671914  2.342040
##   0.00    3    2.602614  0.7402696  2.054916
##   0.00    4    2.429008  0.7887525  1.911864
##   0.00    5    2.942647  0.6793201  2.312460
##   0.00    6    6.049901  0.5165524  3.456843
##   0.00    7    4.082847  0.5712517  2.767596
##   0.00    8    5.767071  0.4572169  3.729774
##   0.00    9    9.810918  0.4053252  5.015810
##   0.00   10    8.474575  0.4636948  4.189643
##   0.01    1    2.661283  0.7278718  2.088473
##   0.01    2    2.852317  0.6877988  2.221027
##   0.01    3    2.523993  0.7568869  2.093470
##   0.01    4    2.418698  0.7832625  1.919667
##   0.01    5    2.698246  0.7446424  2.145357
##   0.01    6    2.933846  0.6993014  2.257254
##   0.01    7    3.137988  0.6529993  2.474540
##   0.01    8    3.618008  0.6097143  2.790248
##   0.01    9    3.425052  0.6185905  2.696058
##   0.01   10    3.946134  0.5420892  3.133259
##   0.10    1    2.656159  0.7286333  2.079465
##   0.10    2    2.779733  0.7025162  2.247186
##   0.10    3    2.579335  0.7391527  2.092155
##   0.10    4    2.268440  0.8176035  1.777193
##   0.10    5    2.482582  0.7668027  2.082117
##   0.10    6    2.726439  0.7433608  2.183655
##   0.10    7    2.705609  0.7327884  2.184070
##   0.10    8    3.254073  0.6557975  2.520924
##   0.10    9    3.303520  0.6246589  2.612321
##   0.10   10    2.980404  0.6845555  2.374350
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 4 and decay = 0.1.
```

```r
nnetPred <- predict(nnetTune, newdata = testData$x)
## The function 'postResample can be used to get the test set
## performance values
nnetResult <- data.frame(as.list(postResample(pred = nnetPred, obs = testData$y))) |>
  mutate(model = 'nnet') |>
  relocate(model, RMSE, Rsquared, MAE)
nnetResult
```

```
##   model     RMSE  Rsquared     MAE
## 1  nnet 2.316673 0.7890649 1.81435
```

MARS: Multivariate Adaptive Regression Splines

```r
marsTune <- train(trainingData$x, trainingData$y,
                  method = 'earth',
                  tuneGrid = expand.grid(.degree = 1:2, .nprune = 2:38),
```

```r
                   trControl = trainControl(method = 'cv'))
marsTune
```

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      4.281001  0.2773318  3.5676969
##   1        3      3.212742  0.6035509  2.6192200
##   1        4      2.677792  0.7274907  2.2467721
##   1        5      2.274281  0.8051169  1.8439150
##   1        6      2.231251  0.8163265  1.8074151
##   1        7      1.800833  0.8717639  1.3852792
##   1        8      1.734264  0.8817659  1.3491174
##   1        9      1.737660  0.8813393  1.3632600
##   1       10      1.764341  0.8770420  1.3938862
##   1       11      1.782250  0.8745491  1.3931766
##   1       12      1.762026  0.8783310  1.3801556
##   1       13      1.742254  0.8836373  1.3674091
##   1       14      1.730718  0.8861096  1.3603181
##   1       15      1.727119  0.8861983  1.3645861
##   1       16      1.724136  0.8870126  1.3614281
##   1       17      1.724136  0.8870126  1.3614281
##   1       18      1.724136  0.8870126  1.3614281
##   1       19      1.724136  0.8870126  1.3614281
##   1       20      1.724136  0.8870126  1.3614281
##   1       21      1.724136  0.8870126  1.3614281
##   1       22      1.724136  0.8870126  1.3614281
##   1       23      1.724136  0.8870126  1.3614281
##   1       24      1.724136  0.8870126  1.3614281
##   1       25      1.724136  0.8870126  1.3614281
##   1       26      1.724136  0.8870126  1.3614281
##   1       27      1.724136  0.8870126  1.3614281
##   1       28      1.724136  0.8870126  1.3614281
##   1       29      1.724136  0.8870126  1.3614281
##   1       30      1.724136  0.8870126  1.3614281
##   1       31      1.724136  0.8870126  1.3614281
##   1       32      1.724136  0.8870126  1.3614281
##   1       33      1.724136  0.8870126  1.3614281
##   1       34      1.724136  0.8870126  1.3614281
##   1       35      1.724136  0.8870126  1.3614281
##   1       36      1.724136  0.8870126  1.3614281
##   1       37      1.724136  0.8870126  1.3614281
##   1       38      1.724136  0.8870126  1.3614281
##   2        2      4.281001  0.2773318  3.5676969
##   2        3      3.212742  0.6035509  2.6192200
```

```
##   2        4      2.677792  0.7274907  2.2467721
##   2        5      2.274281  0.8051169  1.8439150
##   2        6      2.297519  0.8011481  1.8703041
##   2        7      1.837363  0.8665463  1.4149267
##   2        8      1.763882  0.8765845  1.3038700
##   2        9      1.512388  0.9105001  1.1956290
##   2       10      1.329770  0.9296692  1.0481585
##   2       11      1.304416  0.9324248  1.0315848
##   2       12      1.258418  0.9376861  1.0173354
##   2       13      1.269542  0.9369509  1.0247277
##   2       14      1.314616  0.9331632  1.0613778
##   2       15      1.263303  0.9382605  1.0125270
##   2       16      1.234765  0.9412279  0.9839301
##   2       17      1.222248  0.9426109  0.9731562
##   2       18      1.207101  0.9438281  0.9572367
##   2       19      1.211509  0.9434808  0.9567834
##   2       20      1.214652  0.9430248  0.9615431
##   2       21      1.214652  0.9430248  0.9615431
##   2       22      1.214652  0.9430248  0.9615431
##   2       23      1.214652  0.9430248  0.9615431
##   2       24      1.214652  0.9430248  0.9615431
##   2       25      1.214652  0.9430248  0.9615431
##   2       26      1.214652  0.9430248  0.9615431
##   2       27      1.214652  0.9430248  0.9615431
##   2       28      1.214652  0.9430248  0.9615431
##   2       29      1.214652  0.9430248  0.9615431
##   2       30      1.214652  0.9430248  0.9615431
##   2       31      1.214652  0.9430248  0.9615431
##   2       32      1.214652  0.9430248  0.9615431
##   2       33      1.214652  0.9430248  0.9615431
##   2       34      1.214652  0.9430248  0.9615431
##   2       35      1.214652  0.9430248  0.9615431
##   2       36      1.214652  0.9430248  0.9615431
##   2       37      1.214652  0.9430248  0.9615431
##   2       38      1.214652  0.9430248  0.9615431
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 18 and degree = 2.
```

```r
marsPred <- predict(marsTune, newdata = testData$x)
marsResult <- data.frame(as.list(postResample(pred = marsPred, obs = testData$y))) |>
  mutate(model = 'MARS') |>
  relocate(model, RMSE, Rsquared, MAE)
marsResult
```

```
##   model     RMSE  Rsquared       MAE
## 1  MARS 1.158605 0.9463841 0.9242767
```

SVM: Support Vector Machines

```r
svmTune <- train(trainingData$x, trainingData$y,
                 method = 'svmRadial',
```

```r
                 preProcess = c('center','scale'),
                 tuneLength = 14,
                 trControl = trainControl(method = 'cv'))
svmTune
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   C        RMSE      Rsquared   MAE
##      0.25  2.820460  0.7485044  2.278719
##      0.50  2.564032  0.7687252  2.035972
##      1.00  2.385066  0.7912188  1.868641
##      2.00  2.235240  0.8126255  1.739319
##      4.00  2.154486  0.8242148  1.693301
##      8.00  2.097949  0.8332023  1.657564
##     16.00  2.055998  0.8401903  1.634208
##     32.00  2.053345  0.8405006  1.632027
##     64.00  2.053345  0.8405006  1.632027
##    128.00  2.053345  0.8405006  1.632027
##    256.00  2.053345  0.8405006  1.632027
##    512.00  2.053345  0.8405006  1.632027
##   1024.00  2.053345  0.8405006  1.632027
##   2048.00  2.053345  0.8405006  1.632027
##
## Tuning parameter 'sigma' was held constant at a value of 0.0645588
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.0645588 and C = 32.
```

```r
svmPred <- predict(svmTune, testData$x)
svmResult <- data.frame(as.list(postResample(pred = svmPred, obs = testData$y))) |>
  mutate(model = 'SVM') |>
  relocate(model, RMSE, Rsquared, MAE)
svmResult
```

```
##   model     RMSE  Rsquared      MAE
## 1   SVM 1.923757 0.8504649 1.509494
```

Summary

Which models appear to give the best performance?

```r
knnResult |>
  union(nnetResult) |>
  union(marsResult) |>
  union(svmResult) |>
  arrange(desc(Rsquared))
```

```
##   model     RMSE  Rsquared       MAE
## 1  MARS 1.158605 0.9463841 0.9242767
## 2   SVM 1.923757 0.8504649 1.5094939
## 3  nnet 2.316673 0.7890649 1.8143496
## 4   knn 3.241683 0.6701753 2.6008787
```

The MARS model appears to give the best performance based on the RMSE and R^2 statistics.

Does MARS select the informative predictors (those named `X1-X5`)?

```r
varImp(marsTune)$importance |>
  arrange(desc(Overall)) |>
  head(10)
```

```
##      Overall
## X4 100.00000
## X2  71.30508
## X1  37.78989
## X5  15.48104
## X3   0.00000
```

The MARS model selects the informative predictors, but `X3` appears to be insignificant and has an overall importance of 0.

Exercise 7.5

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 nonlinear regression models.

From Homework 7

```r
data(ChemicalManufacturingProcess)
imputed <- predict(preProcess(ChemicalManufacturingProcess, method = 'bagImpute'), ChemicalManufacturing

X <- imputed |>
  select(-Yield)
y <- imputed$Yield

X <- X[,-nearZeroVar(X)]

train <- createDataPartition(y, p = .8, list = FALSE)
X_train <- X[train,]
X_test <- X[-train,]
y_train <- y[train]
y_test <- y[-train]
```

KNN: k-Nearest Neighbors

```r
knnModel <- train(X_train, y_train,
                  method = 'knn',
                  preProcess = c('center','scale'),
                  tuneLength = 10)
knnModel
```

```
## k-Nearest Neighbors
##
## 144 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 144, 144, 144, 144, 144, 144, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  1.379658  0.4505493  1.090764
##    7  1.384271  0.4461378  1.110260
##    9  1.396146  0.4379598  1.121155
##   11  1.417544  0.4222474  1.141312
##   13  1.429682  0.4144088  1.149497
##   15  1.441541  0.4051353  1.160162
##   17  1.452704  0.3976862  1.171167
##   19  1.460561  0.3943526  1.178263
##   21  1.467243  0.3907426  1.185358
##   23  1.470348  0.3912654  1.183060
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.
```

```r
knnPred <- predict(knnModel, newdata = X_test)
knnResult <- data.frame(as.list(postResample(pred = knnPred, obs = y_test))) |>
  mutate(model = 'knn') |>
  relocate(model, RMSE, Rsquared, MAE)
knnResult
```

```
##   model     RMSE  Rsquared      MAE
## 1   knn 1.498845 0.3961932 1.179875
```

NNET: Neural Networks

```r
nnetModel <- train(X_train, y_train,
                  method = 'nnet',
                  tuneGrid = expand.grid(.decay = c(0, 0.01, .1), .size = c(1:10)),
                  trControl = trainControl(method = 'cv', number = 10),
                  preProcess = c('center', 'scale'),
                  linout = TRUE, trace = FALSE,
                  MaxNWts = 10 * (ncol(X_train) + 1) + 10 + 1,
                  maxit = 500)
nnetModel
```

```
## Neural Network
##
## 144 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 129, 129, 132, 130, 128, 129, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE       Rsquared    MAE
##   0.00    1     1.678361  0.23432174  1.338730
##   0.00    2     1.556209  0.29289498  1.250798
##   0.00    3     3.061613  0.23730838  2.252601
##   0.00    4     3.912098  0.10612415  3.005222
##   0.00    5     4.071541  0.07447244  3.229146
##   0.00    6     4.083318  0.16970212  3.204658
##   0.00    7     5.641098  0.15745984  4.261070
##   0.00    8     6.831032  0.11623219  5.146468
##   0.00    9     6.888511  0.06602355  5.191935
##   0.00   10    10.884295  0.18549679  8.121485
##   0.01    1     1.823209  0.30410158  1.507439
##   0.01    2     2.396530  0.26077806  1.972948
##   0.01    3     2.409577  0.26202087  1.921315
##   0.01    4     2.518533  0.28321867  1.988295
##   0.01    5     2.907159  0.16737936  2.228059
##   0.01    6     2.748951  0.17285032  2.146255
##   0.01    7     2.875025  0.15816481  2.092122
##   0.01    8     2.558987  0.18372653  1.964162
##   0.01    9     2.643811  0.27458246  2.051323
##   0.01   10     3.760922  0.20555320  2.709466
##   0.10    1     1.882574  0.36482588  1.355045
##   0.10    2     2.086595  0.28868348  1.648466
##   0.10    3     2.570318  0.20776660  1.966156
##   0.10    4     2.372222  0.30209195  1.821054
##   0.10    5     2.811116  0.27924167  2.069933
##   0.10    6     2.250491  0.24316559  1.726597
##   0.10    7     2.595976  0.17133799  1.872720
##   0.10    8     2.705299  0.21229635  2.055154
##   0.10    9     2.716765  0.16406457  2.073858
##   0.10   10     2.247612  0.25337782  1.856046
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 2 and decay = 0.
```

```r
nnetPred <- predict(nnetModel, newdata = X_test)
nnetResult <- data.frame(as.list(postResample(pred = nnetPred, obs = y_test))) |>
  mutate(model = 'nnet') |>
  relocate(model, RMSE, Rsquared, MAE)
nnetResult
```

```
##   model     RMSE  Rsquared      MAE
## 1  nnet 1.630832 0.2647858 1.388332
```

MARS: Multivariate Adaptive Regression Splines

```r
marsModel <- train(X_train, y_train,
                   method = 'earth',
                   tuneGrid = expand.grid(.degree = 1:2, .nprune = 2:38),
                   trControl = trainControl(method = 'cv'))
marsModel
```

```
## Multivariate Adaptive Regression Spline
##
## 144 samples
##  56 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 131, 128, 129, 130, 130, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      1.433126  0.4154294  1.1216934
##   1        3      1.276615  0.5137493  1.0293659
##   1        4      1.223836  0.5580225  1.0041552
##   1        5      1.232831  0.5532128  0.9990345
##   1        6      1.216539  0.5638381  0.9751419
##   1        7      1.202638  0.5729486  0.9357264
##   1        8      1.166423  0.5908522  0.9340754
##   1        9      1.144488  0.6014037  0.9115922
##   1       10      1.174225  0.5804208  0.9336118
##   1       11      1.184131  0.5714902  0.9418250
##   1       12      1.161849  0.5937193  0.9311139
##   1       13      1.154729  0.6037815  0.9077335
##   1       14      1.128364  0.6188913  0.8886522
##   1       15      1.126799  0.6207559  0.8870071
##   1       16      1.136196  0.6182286  0.8955953
##   1       17      1.132228  0.6207677  0.8914600
##   1       18      1.135646  0.6180451  0.8971810
##   1       19      1.135646  0.6180451  0.8971810
##   1       20      1.135646  0.6180451  0.8971810
##   1       21      1.135646  0.6180451  0.8971810
##   1       22      1.135646  0.6180451  0.8971810
##   1       23      1.135646  0.6180451  0.8971810
##   1       24      1.135646  0.6180451  0.8971810
##   1       25      1.135646  0.6180451  0.8971810
##   1       26      1.135646  0.6180451  0.8971810
##   1       27      1.135646  0.6180451  0.8971810
##   1       28      1.135646  0.6180451  0.8971810
##   1       29      1.135646  0.6180451  0.8971810
##   1       30      1.135646  0.6180451  0.8971810
##   1       31      1.135646  0.6180451  0.8971810
##   1       32      1.135646  0.6180451  0.8971810
##   1       33      1.135646  0.6180451  0.8971810
##   1       34      1.135646  0.6180451  0.8971810
##   1       35      1.135646  0.6180451  0.8971810
```

```
##    1      36     1.135646  0.6180451  0.8971810
##    1      37     1.135646  0.6180451  0.8971810
##    1      38     1.135646  0.6180451  0.8971810
##    2       2     1.433126  0.4154294  1.1216934
##    2       3     1.406667  0.4303385  1.1173924
##    2       4     1.243403  0.5446763  1.0055262
##    2       5     1.277720  0.5231206  1.0168204
##    2       6     1.337607  0.5119702  1.0407807
##    2       7     1.362614  0.4895329  1.0644624
##    2       8     1.303091  0.5152020  1.0262146
##    2       9     1.325853  0.5117465  1.0355350
##    2      10     1.372537  0.5081267  1.0853983
##    2      11     1.360297  0.5046625  1.0785989
##    2      12     1.384471  0.4917035  1.0801942
##    2      13     1.382711  0.4893548  1.0805039
##    2      14     1.433577  0.4586557  1.1181035
##    2      15     1.423909  0.4867217  1.1160937
##    2      16     1.449919  0.4700785  1.1314275
##    2      17     1.447589  0.4778954  1.1246925
##    2      18     1.441212  0.4831453  1.1186850
##    2      19     1.461453  0.4704108  1.1359042
##    2      20     1.431758  0.4888435  1.1183069
##    2      21     1.388058  0.5113556  1.0878968
##    2      22     1.393505  0.5100796  1.0920114
##    2      23     1.403067  0.5134808  1.1018171
##    2      24     1.431915  0.5054626  1.1227629
##    2      25     1.433135  0.5052046  1.1194694
##    2      26     1.444916  0.5082111  1.1303573
##    2      27     1.444916  0.5082111  1.1303573
##    2      28     1.444916  0.5082111  1.1303573
##    2      29     1.444916  0.5082111  1.1303573
##    2      30     1.444916  0.5082111  1.1303573
##    2      31     1.444916  0.5082111  1.1303573
##    2      32     1.444916  0.5082111  1.1303573
##    2      33     1.444916  0.5082111  1.1303573
##    2      34     1.444916  0.5082111  1.1303573
##    2      35     1.444916  0.5082111  1.1303573
##    2      36     1.444916  0.5082111  1.1303573
##    2      37     1.444916  0.5082111  1.1303573
##    2      38     1.444916  0.5082111  1.1303573
##
## 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.
```

```r
marsPred <- predict(marsModel, newdata = X_test)
marsResult <- data.frame(as.list(postResample(pred = marsPred, obs = y_test))) |>
  mutate(model = 'MARS') |>
  relocate(model, RMSE, Rsquared, MAE)
marsResult
```

```
##   model     RMSE  Rsquared      MAE
## 1  MARS 1.066548 0.6964923 0.796169
```

SVM: Support Vector Machines

```r
svmModel <- train(X_train, y_train,
                  method = 'svmRadial',
                  preProcess = c('center','scale'),
                  tuneLength = 14,
                  trControl = trainControl(method = 'cv'))
svmModel
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 144 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 130, 130, 131, 129, 130, 129, ...
## Resampling results across tuning parameters:
##
##   C        RMSE      Rsquared   MAE
##      0.25  1.412475  0.4832852  1.1352064
##      0.50  1.330770  0.5201218  1.0597256
##      1.00  1.271628  0.5616200  1.0033987
##      2.00  1.225052  0.5966922  0.9705721
##      4.00  1.199345  0.6102707  0.9591661
##      8.00  1.191542  0.6245313  0.9588123
##     16.00  1.188397  0.6264261  0.9553073
##     32.00  1.188397  0.6264261  0.9553073
##     64.00  1.188397  0.6264261  0.9553073
##    128.00  1.188397  0.6264261  0.9553073
##    256.00  1.188397  0.6264261  0.9553073
##    512.00  1.188397  0.6264261  0.9553073
##   1024.00  1.188397  0.6264261  0.9553073
##   2048.00  1.188397  0.6264261  0.9553073
##
## Tuning parameter 'sigma' was held constant at a value of 0.01439803
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.01439803 and C = 16.
```

```r
svmPred <- predict(svmModel, newdata = X_test)
svmResult <- data.frame(as.list(postResample(pred = svmPred, obs = y_test))) |>
  mutate(model = 'SVM') |>
  relocate(model, RMSE, Rsquared, MAE)
svmResult
```

```
##   model     RMSE  Rsquared       MAE
## 1   SVM 1.074226 0.7380873 0.8590428
```

Summary

a. Which nonlinear regression model gives the optimal resampling and test set performance?

```r
knnResult |>
  union(nnetResult) |>
  union(marsResult) |>
  union(svmResult) |>
  arrange(desc(Rsquared))
```

```
##   model     RMSE  Rsquared       MAE
## 1   SVM 1.074226 0.7380873 0.8590428
## 2  MARS 1.066548 0.6964923 0.7961690
## 3   knn 1.498845 0.3961932 1.1798750
## 4  nnet 1.630832 0.2647858 1.3883322
```

The SVM model produced the highest R^2 value indicating it is the best model.

b. Which predictors are most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

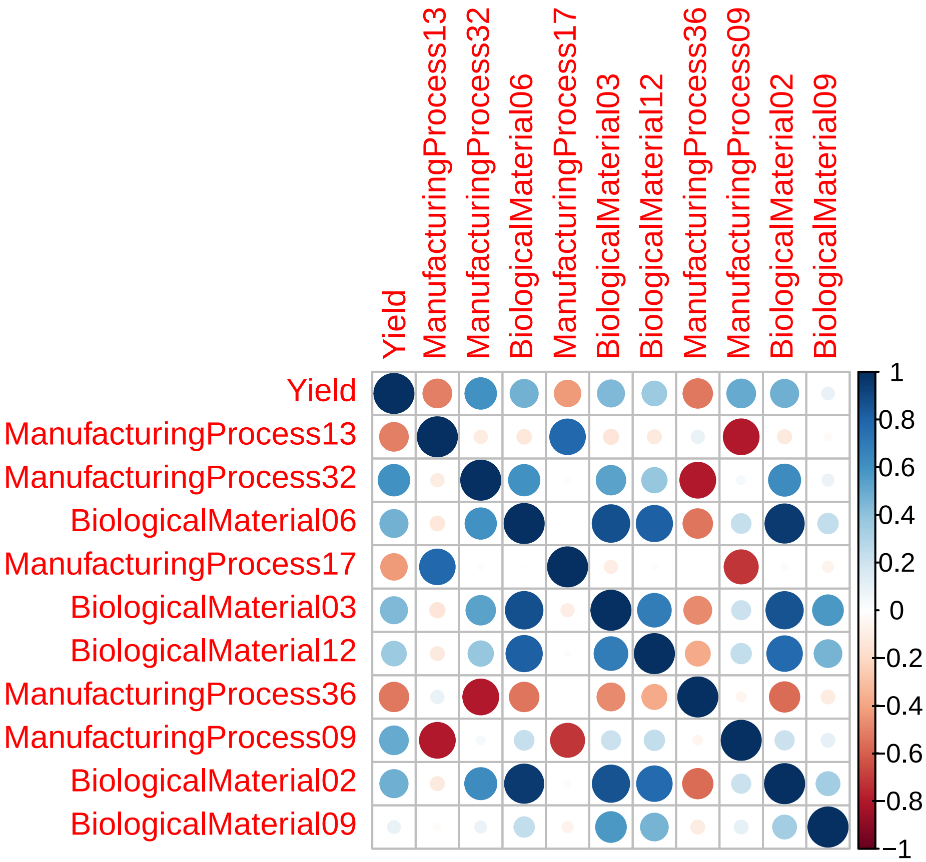
```r
top10 <- varImp(svmModel)$importance |>
  arrange(desc(Overall)) |>
  head(10)
top10
```

```
##                          Overall
## ManufacturingProcess13 100.00000
## ManufacturingProcess32  97.21774
## BiologicalMaterial06    86.44833
## ManufacturingProcess17  84.61263
## BiologicalMaterial03    82.57291
## BiologicalMaterial12    81.65567
## ManufacturingProcess36  72.50145
## ManufacturingProcess09  68.96933
## BiologicalMaterial02    65.77812
## BiologicalMaterial09    57.98958
```

The manufacturing process and biological material predictors are split evenly in the top 10 in the SVM model. In the previous exercise, the PLS model was the best linear model and the manufacturing processes were the most important predictors.

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

```r
imputed |>
  select(c('Yield', row.names(top10))) |>
  cor() |>
  corrplot::corrplot()
```



Manufacturing Process 32 and 09 have the strongest correlation with the Yield and Manufacturing Process 36 has the highest inverse correlation with the Yield.