DATA 624 Homework 8 - Non-Linear Regression

Kevin Kirby

2024-11-01

Overview

This is homework eight of the Fall 2024 edition of DATA 624. The assignment covers questions 7.2 and 7.5 from the exercise section of chapter 7 in Applied Predictive Modeling by Max Kuhn and Kjell Johnson

First, most of the requried libraries

```
library(tidyverse)
library(ggplot2)
library(gridExtra)
library(AppliedPredictiveModeling)
library(mlbench)
library(caret)
library(earth)
library(kernlab)
library(RANN)


set.seed(04101)
```

7.2 Tuning models on benchmmark datasets:

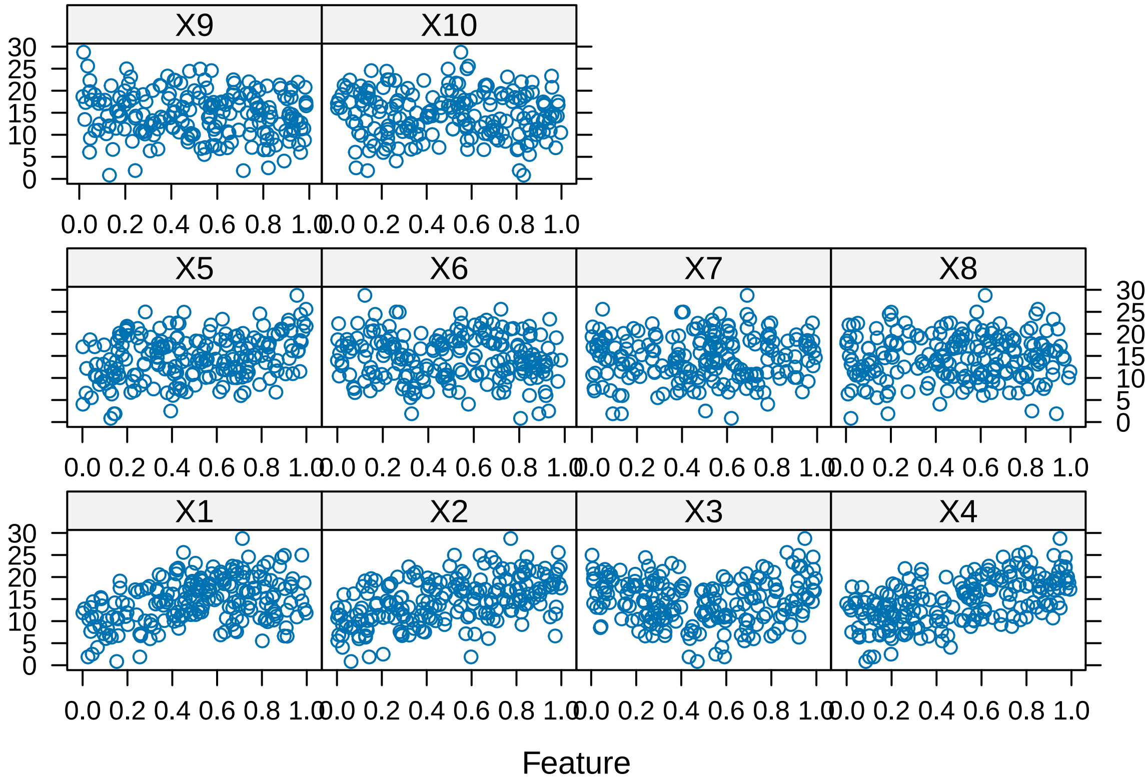
The exercise states: "Friedman (1991) introduced several benchmark data sets create by simulation. One of these simulations used the following nonlinear equation to create data:

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

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"

The following code snippet was also provided:

```
trainingData <- mlbench.friedman1(200, sd = 1)
trainingData$x <- data.frame(trainingData$x)
featurePlot(trainingData$x, trainingData$y)
```




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

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  3.719126  0.4938384  2.959920
##    7  3.596913  0.5402103  2.837807
##    9  3.604542  0.5501806  2.850472
##   11  3.612526  0.5576494  2.867785
##   13  3.606714  0.5717275  2.882308
##   15  3.625217  0.5791756  2.909228
##   17  3.610499  0.5969979  2.901709
##   19  3.606544  0.6127054  2.903831
```

```
##   21  3.611499  0.6263947  2.912113
##   23  3.629909  0.6367596  2.934772
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

I must now tune "several models on these data," and then provide comentary. I ave chosen the following models:

- Multivariate Adaptive Regression Splines (MARS)
- Neural Networks
- Support Vector Machines

7.2 - Multivariate Adaptive Regression Splines

I'm starting with because it's what I'm presenting on during my team's presentation. MARS is a nonparametric regression technique that allows you to model relationships without assuming a particular shape of relationship at the start It's a piecewise model with defined linear relationships called "splines" what map different segments of the data. The boundaries of the segments can overlap and are tied together with "knots.'

```r
mc <- trainControl(method = "repeatedcv", number = 3)
mexpand_g <- expand.grid(.degree = 2:3, .nprune = 4:45)

life_on_mars <- train(trainingData$x, trainingData$y,
                      method = "earth",
                      tuneGrid = mexpand_g,
                      preProcess = c("center", "scale", "knnImpute"),
                      tuneLength = 15,
                      trControl = mc)

life_on_mars
```

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10), nearest neighbor imputation (10)
## Resampling: Cross-Validated (3 fold, repeated 1 times)
## Summary of sample sizes: 134, 133, 133
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   2        4      3.311659  0.6019001  2.719880
##   2        5      3.132180  0.6448154  2.545251
##   2        6      2.912083  0.6896125  2.360018
##   2        7      2.444828  0.7788718  1.902028
##   2        8      2.272775  0.8110137  1.744256
##   2        9      1.944770  0.8591212  1.464891
##   2       10      1.665242  0.8985534  1.324536
##   2       11      1.635008  0.9012180  1.314050
##   2       12      1.427742  0.9253497  1.153250
##   2       13      1.522163  0.9154320  1.216036
##   2       14      1.562438  0.9109944  1.238524
##   2       15      1.552673  0.9114834  1.250201
##   2       16      1.535044  0.9133345  1.235728
```

```
## 2    17  1.535044 0.9133345 1.235728
## 2    18  1.535044 0.9133345 1.235728
## 2    19  1.535044 0.9133345 1.235728
## 2    20  1.535044 0.9133345 1.235728
## 2    21  1.535044 0.9133345 1.235728
## 2    22  1.535044 0.9133345 1.235728
## 2    23  1.535044 0.9133345 1.235728
## 2    24  1.535044 0.9133345 1.235728
## 2    25  1.535044 0.9133345 1.235728
## 2    26  1.535044 0.9133345 1.235728
## 2    27  1.535044 0.9133345 1.235728
## 2    28  1.535044 0.9133345 1.235728
## 2    29  1.535044 0.9133345 1.235728
## 2    30  1.535044 0.9133345 1.235728
## 2    31  1.535044 0.9133345 1.235728
## 2    32  1.535044 0.9133345 1.235728
## 2    33  1.535044 0.9133345 1.235728
## 2    34  1.535044 0.9133345 1.235728
## 2    35  1.535044 0.9133345 1.235728
## 2    36  1.535044 0.9133345 1.235728
## 2    37  1.535044 0.9133345 1.235728
## 2    38  1.535044 0.9133345 1.235728
## 2    39  1.535044 0.9133345 1.235728
## 2    40  1.535044 0.9133345 1.235728
## 2    41  1.535044 0.9133345 1.235728
## 2    42  1.535044 0.9133345 1.235728
## 2    43  1.535044 0.9133345 1.235728
## 2    44  1.535044 0.9133345 1.235728
## 2    45  1.535044 0.9133345 1.235728
## 3     4  3.311659 0.6019001 2.719880
## 3     5  3.132180 0.6448154 2.545251
## 3     6  2.912083 0.6896125 2.360018
## 3     7  2.444828 0.7788718 1.902028
## 3     8  2.272775 0.8110137 1.744256
## 3     9  1.944770 0.8591212 1.464891
## 3    10  1.665242 0.8985534 1.324536
## 3    11  1.635008 0.9012180 1.314050
## 3    12  1.427742 0.9253497 1.153250
## 3    13  1.522163 0.9154320 1.216036
## 3    14  1.562438 0.9109944 1.238524
## 3    15  1.552673 0.9114834 1.250201
## 3    16  1.535044 0.9133345 1.235728
## 3    17  1.535044 0.9133345 1.235728
## 3    18  1.535044 0.9133345 1.235728
## 3    19  1.535044 0.9133345 1.235728
## 3    20  1.535044 0.9133345 1.235728
## 3    21  1.535044 0.9133345 1.235728
## 3    22  1.535044 0.9133345 1.235728
## 3    23  1.535044 0.9133345 1.235728
## 3    24  1.535044 0.9133345 1.235728
## 3    25  1.535044 0.9133345 1.235728
## 3    26  1.535044 0.9133345 1.235728
## 3    27  1.535044 0.9133345 1.235728
## 3    28  1.535044 0.9133345 1.235728
```

```
## 3 29 1.535044 0.9133345 1.235728
## 3 30 1.535044 0.9133345 1.235728
## 3 31 1.535044 0.9133345 1.235728
## 3 32 1.535044 0.9133345 1.235728
## 3 33 1.535044 0.9133345 1.235728
## 3 34 1.535044 0.9133345 1.235728
## 3 35 1.535044 0.9133345 1.235728
## 3 36 1.535044 0.9133345 1.235728
## 3 37 1.535044 0.9133345 1.235728
## 3 38 1.535044 0.9133345 1.235728
## 3 39 1.535044 0.9133345 1.235728
## 3 40 1.535044 0.9133345 1.235728
## 3 41 1.535044 0.9133345 1.235728
## 3 42 1.535044 0.9133345 1.235728
## 3 43 1.535044 0.9133345 1.235728
## 3 44 1.535044 0.9133345 1.235728
## 3 45 1.535044 0.9133345 1.235728
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 12 and degree = 2.
```

```
m_predict <- predict(life_on_mars, newdata = testData$x)
predict_track <- data.frame(matrix(vector(), 0, 4,  # Increased to 4 for the model name
                dimnames = list(NULL, c("RMSE", "r2", "MAE", "model"))),
                stringsAsFactors = FALSE)

m_metrics <- as.data.frame(t(postResample(pred = m_predict, obs = testData$y)))
m_metrics$model <- "MARS"

predict_track <- rbind(predict_track, m_metrics)
```

7.2 - Support Vector Machines

Support Vector Machines finds an optimal hyperplane to separate data points by maximizing the margin between different targets values. A radial basis function can be used for non-linear data as well.

```
svc <- trainControl(method = "repeatedcv", number = 5, repeats = 3)

svc_t <- expand.grid(.C = seq(0.1, 1, by = 0.1), .sigma = seq(0.05, 0.2, by = 0.05))

svm_m <- train(x = trainingData$x,
                  y = trainingData$y,
                  method = "svmRadial",
                  preProcess = c("center", "scale"),
                  tuneGrid = svc_t,
                  trControl = svc)

svm_m
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
```

```
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 160, 160, 160, 160, 160, 160, ...
## Resampling results across tuning parameters:
##
##   C    sigma  RMSE      Rsquared   MAE
##   0.1  0.05   3.723903  0.7106557  3.026526
##   0.1  0.10   3.917585  0.7009434  3.174456
##   0.1  0.15   4.250298  0.6803521  3.453626
##   0.1  0.20   4.530993  0.6483326  3.702109
##   0.2  0.05   3.094990  0.7290384  2.453913
##   0.2  0.10   3.232816  0.7272018  2.559308
##   0.2  0.15   3.630465  0.7016895  2.899013
##   0.2  0.20   4.035662  0.6664100  3.248765
##   0.3  0.05   2.850083  0.7420247  2.197422
##   0.3  0.10   2.956281  0.7441759  2.276565
##   0.3  0.15   3.288211  0.7165915  2.572111
##   0.3  0.20   3.683004  0.6820470  2.921419
##   0.4  0.05   2.735546  0.7503461  2.082240
##   0.4  0.10   2.819071  0.7504706  2.136744
##   0.4  0.15   3.121262  0.7226468  2.400362
##   0.4  0.20   3.481744  0.6898263  2.726998
##   0.5  0.05   2.655809  0.7570843  2.006147
##   0.5  0.10   2.723461  0.7574490  2.048870
##   0.5  0.15   3.017949  0.7254601  2.301119
##   0.5  0.20   3.368566  0.6912825  2.610398
##   0.6  0.05   2.598904  0.7623278  1.959341
##   0.6  0.10   2.661616  0.7623015  2.000087
##   0.6  0.15   2.946295  0.7281582  2.238830
##   0.6  0.20   3.288601  0.6911002  2.527406
##   0.7  0.05   2.549073  0.7682990  1.920879
##   0.7  0.10   2.618822  0.7654494  1.968101
##   0.7  0.15   2.893491  0.7306077  2.198658
##   0.7  0.20   3.227760  0.6925924  2.473100
##   0.8  0.05   2.507235  0.7738224  1.891699
##   0.8  0.10   2.586948  0.7678633  1.945064
##   0.8  0.15   2.855315  0.7331353  2.169950
##   0.8  0.20   3.182047  0.6940005  2.435306
##   0.9  0.05   2.475813  0.7780461  1.870435
##   0.9  0.10   2.561451  0.7701869  1.923824
##   0.9  0.15   2.825430  0.7358589  2.145652
##   0.9  0.20   3.148122  0.6953027  2.409639
##   1.0  0.05   2.449445  0.7815835  1.850789
##   1.0  0.10   2.541630  0.7724449  1.907116
##   1.0  0.15   2.799129  0.7386363  2.124024
##   1.0  0.20   3.120230  0.6971721  2.386707
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.05 and C = 1.
```

```r
svm_p<- predict(svm_m, newdata = testData$x)
svm_metrics <- as.data.frame(t(postResample(pred = svm_p, obs = testData$y)))
svm_metrics$model <- "SVMs"
predict_track <- rbind(predict_track, svm_metrics)
```

Thus far, the The MARS model is ahead of the SVM, with a higher RMSE of 1.34 and R-squared of ~0.938.

For the SVM, the increasing C improved results, but gains leveled off after C = 1. Both models are strong, but MARS might be the better pick here for capturing data variance.

7.2 Neural Networks

This code trains an averaged neural network model using the avNNet method, applying preprocessing steps to center and scale the data. It customizes the model with a tuning grid, cross-validation controls, and specific parameters for network size, weights, and iteration limits.

```r
nnet_grid <- expand.grid(.decay = c(0, 0.001, 0.01, 0.05, 0.1), .size = 1:15, .bag = FALSE)
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

nnet_maxnwts <- 150
nnet_model <- train(x = trainingData$x,
                    y = trainingData$y,
                    method = "avNNet",
                    preProcess = c("center", "scale"),
                    tuneGrid = nnet_grid,
                    trControl = control,
                    linout = TRUE,
                    trace = FALSE,
                    MaxNWts = nnet_maxnwts,
                    maxit = 300)

nnet_model
```

```
## Model Averaged Neural Network
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE      Rsquared   MAE
##   0.000   1    2.802094  0.7190798  2.204010
##   0.000   2    2.523775  0.7718422  2.000498
##   0.000   3    2.121372  0.8332285  1.695636
##   0.000   4    2.013886  0.8546191  1.567829
##   0.000   5    2.367556  0.8071777  1.787363
##   0.000   6    3.162169  0.7048747  2.247950
##   0.000   7    4.834655  0.5205897  3.024356
##   0.000   8    5.474489  0.5289465  3.301518
##   0.000   9    4.923984  0.5868559  2.905196
##   0.000  10    3.849287  0.6118177  2.563422
##   0.000  11    3.334532  0.6533816  2.500506
##   0.000  12    3.461783  0.6910991  2.596698
##   0.000  13         NaN        NaN       NaN
##   0.000  14         NaN        NaN       NaN
##   0.000  15         NaN        NaN       NaN
##   0.001   1    2.778055  0.7207142  2.177187
##   0.001   2    2.539277  0.7694598  2.012517
##   0.001   3    2.201545  0.8238616  1.732125
```

```
##    0.001   4    2.038469  0.8522219  1.612823
##    0.001   5    2.222230  0.8204783  1.728492
##    0.001   6    2.461068  0.7799747  1.910654
##    0.001   7    2.766955  0.7389167  2.167112
##    0.001   8    3.009032  0.7142539  2.245309
##    0.001   9    3.368694  0.6685557  2.542988
##    0.001  10    2.924293  0.7198047  2.257227
##    0.001  11    2.878914  0.7230730  2.244031
##    0.001  12    2.859150  0.7165688  2.268190
##    0.001  13         NaN        NaN       NaN
##    0.001  14         NaN        NaN       NaN
##    0.001  15         NaN        NaN       NaN
##    0.010   1    2.756020  0.7256414  2.160956
##    0.010   2    2.532561  0.7679450  2.032686
##    0.010   3    2.219390  0.8185134  1.753755
##    0.010   4    2.098196  0.8435576  1.643065
##    0.010   5    2.165545  0.8311878  1.676269
##    0.010   6    2.367029  0.7981274  1.837282
##    0.010   7    2.526360  0.7748758  1.962880
##    0.010   8    2.743666  0.7396926  2.116238
##    0.010   9    2.838719  0.7198539  2.190079
##    0.010  10    2.751071  0.7282082  2.156551
##    0.010  11    2.867661  0.7150112  2.305827
##    0.010  12    2.946496  0.6983125  2.324035
##    0.010  13         NaN        NaN       NaN
##    0.010  14         NaN        NaN       NaN
##    0.010  15         NaN        NaN       NaN
##    0.050   1    2.728565  0.7295766  2.136572
##    0.050   2    2.550234  0.7664020  2.020062
##    0.050   3    2.243660  0.8151612  1.767740
##    0.050   4    2.066508  0.8456286  1.623204
##    0.050   5    2.195013  0.8246665  1.705472
##    0.050   6    2.385456  0.7988194  1.871967
##    0.050   7    2.516904  0.7742450  1.939992
##    0.050   8    2.612743  0.7627249  2.030529
##    0.050   9    2.704107  0.7438953  2.113998
##    0.050  10    2.621658  0.7584304  2.045466
##    0.050  11    2.730381  0.7400997  2.144502
##    0.050  12    2.788834  0.7317214  2.178176
##    0.050  13         NaN        NaN       NaN
##    0.050  14         NaN        NaN       NaN
##    0.050  15         NaN        NaN       NaN
##    0.100   1    2.736295  0.7278178  2.142121
##    0.100   2    2.527389  0.7711798  2.028058
##    0.100   3    2.168856  0.8232641  1.733253
##    0.100   4    2.155233  0.8333829  1.701868
##    0.100   5    2.172664  0.8277510  1.698866
##    0.100   6    2.291981  0.8124337  1.796571
##    0.100   7    2.508576  0.7766792  1.973119
##    0.100   8    2.534458  0.7659478  1.964556
##    0.100   9    2.573739  0.7640563  2.003362
##    0.100  10    2.610022  0.7561825  2.029116
##    0.100  11    2.522858  0.7767753  1.991092
##    0.100  12    2.674768  0.7477552  2.086116
```

```
##   0.100  13         NaN        NaN       NaN
##   0.100  14         NaN        NaN       NaN
##   0.100  15         NaN        NaN       NaN
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 4, decay = 0 and bag = FALSE.
```

```
nn_p <- predict(nnet_model, newdata = testData$x)
nn_metrics <- as.data.frame(t(postResample(pred = nn_p, obs = testData$y)))
nn_metrics$model <- "Neural Networks"
predict_track <- rbind(predict_track, nn_metrics)

predict_track
```

```
##       RMSE  Rsquared       MAE           model
## 1 1.233097 0.9382421 0.9869724            MARS
## 2 2.296046 0.7887541 1.7584783            SVMs
## 3 1.931904 0.8493991 1.4735077 Neural Networks
```

The final part of the question states: "Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)?"

Answer:

Based on the above, the mars model came out ahead. It did a better job of capturing the underlying patterns in the data. While SVMs and Neural Networks showed decent performance, their higher RMSE values suggest that they may not predict as accurately as MARS for this particular dataset.

7.5 training several non-linear models

The exercise states: "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.

I'll first bring in the data from 6.3.

```
data("ChemicalManufacturingProcess")
chem_ct <- trainControl(method = "repeatedcv", number = 3)

mexpand_gt <- expand.grid(.degree = 2:3, .nprune = 4:45)


chem_pre <- preProcess(ChemicalManufacturingProcess,
                       method = c("BoxCox", "knnImpute", "center", "scale"))
chem_pred <- predict(chem_pre, ChemicalManufacturingProcess)
chem_pred$Yield = ChemicalManufacturingProcess$Yield

chem_i <- sample(seq_len(nrow(chem_pred)), size = floor(0.85 * nrow(chem_pred)))
ctr <- chem_pred[chem_i, ]
ctt <- chem_pred[-chem_i, ]
```

Now I can run amm the models again, using the same order as earlier.

```
knnm <- train(Yield ~ ., data = ctr,
              method = "knn",
              preProcess = c("center", "scale"),
              tuneLength = 10)
knnpt <- predict(knnm, newdata = ctt)
```

```r
nnet_g <- expand.grid(.decay = c(0, 0.001, 0.01), .size = 1:1, .bag = FALSE)
nnet_mw <- 5 * (ncol(ctr) + 1) + 5 + 1

nnmt <- train(Yield ~ ., data = ctr,
              method = "avNNet",
              tuneGrid = nnet_g,
              trControl = chem_ct,
              linout = TRUE,
              trace = FALSE,
              MaxNWts = nnet_mw,
              maxit = 130)
nnpt <- predict(nnmt, newdata = ctt)

marst <- train(Yield ~ ., data = ctr,
               method = "earth",
               tuneGrid = mexpand_gt,
               trControl = chem_ct)
marspt <- predict(marst, newdata = ctt)

svmtt <- train(Yield ~ ., data = ctr,
               method = "svmRadial",
               tuneLength = 8,
               trControl = chem_ct)
svmpt <- predict(svmtt, newdata = ctt)

chem_r <- data.frame(Model = character(),
                     RMSE = numeric(),
                     R_squared = numeric(),
                     MAE = numeric(),
                     stringsAsFactors = FALSE)

knn_r <- postResample(pred = knnpt, obs = ctt$Yield)
chem_r <- rbind(chem_r, data.frame(Model = "KNN",
                                    RMSE = knn_r[1],
                                    R_squared = knn_r[2],
                                    MAE = knn_r[3]))

nn_r <- postResample(pred = nnpt, obs = ctt$Yield)
chem_r <- rbind(chem_r, data.frame(Model = "Averaged Neural Network",
                                    RMSE = nn_r[1],
                                    R_squared = nn_r[2],
                                    MAE = nn_r[3]))

mars_r <- postResample(pred = marspt, obs = ctt$Yield)
chem_r <- rbind(chem_r, data.frame(Model = "MARS",
                                    RMSE = mars_r[1],
                                    R_squared = mars_r[2],
                                    MAE = mars_r[3]))

svm_results <- postResample(pred = svmpt, obs = ctt$Yield)
chem_r <- rbind(chem_r, data.frame(Model = "SVM",
                                    RMSE = svm_results[1],
                                    R_squared = svm_results[2],
```

```
                                        MAE = svm_results[3]))
chem_r
```

```
##                          Model     RMSE R_squared       MAE
## RMSE                       KNN 1.222443 0.4035379 0.9609687
## RMSE1 Averaged Neural Network 1.204325 0.3933972 0.9887849
## RMSE2                     MARS 1.247842 0.3866294 0.9904590
## RMSE3                      SVM 1.159326 0.4430202 0.9447833
```

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

Answer: SVM has the lowest RMSE at 0.7649 and the highest R-squared at 0.8526684 among all models. This shows it has best predictive accuracy and also accounts for a lot of the variance in the data. It also has the lowest MAE at 0.6, which means its making the smallest average prediction errors as well.

Based on the above, I would say SVM provided the optimal resampling and test set performance based.

```
perfs <- as.data.frame(rbind(
  "KNN" = postResample(pred = knnpt, obs = ctt $Yield),
  "MARS" = postResample(pred = marspt, obs = ctt$Yield),
  "SVM" = postResample(pred = svmpt, obs = ctt $Yield),
  "Averaged Neural Network" = postResample(pred = nnpt, obs = ctt $Yield)
))

perfs <- cbind(Model = rownames(perfs), perfs)
rownames(perfs) <- NULL

perfs <- perfs %>% arrange(RMSE)

perfs
```

```
##                     Model     RMSE  Rsquared       MAE
## 1                     SVM 1.159326 0.4430202 0.9447833
## 2 Averaged Neural Network 1.204325 0.3933972 0.9887849
## 3                     KNN 1.222443 0.4035379 0.9609687
## 4                    MARS 1.247842 0.3866294 0.9904590
```

7.5.b - Determining the most important predictors

The next component question states:

"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?"

Answer: As with the last time we worked with this same dataset, there's pretty much an even split between biological and process variables at the top.

```
varImp(svmtt, 10)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 57)
##
##                        Overall
## ManufacturingProcess32  100.00
## ManufacturingProcess13   89.95
```

```
## BiologicalMaterial06      82.97
## BiologicalMaterial03      78.17
## ManufacturingProcess17    75.61
## ManufacturingProcess36    69.03
## ManufacturingProcess09    66.81
## BiologicalMaterial12      66.17
## ManufacturingProcess31    65.78
## ManufacturingProcess06    63.53
## BiologicalMaterial02      61.89
## ManufacturingProcess33    52.91
## ManufacturingProcess29    49.55
## BiologicalMaterial01      47.04
## BiologicalMaterial04      46.91
## BiologicalMaterial11      43.99
## ManufacturingProcess11    42.92
## BiologicalMaterial08      42.42
## BiologicalMaterial09      40.69
## ManufacturingProcess12    37.30
```

7.5.c Exploring relationships in the data

The final component question states:

"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?"

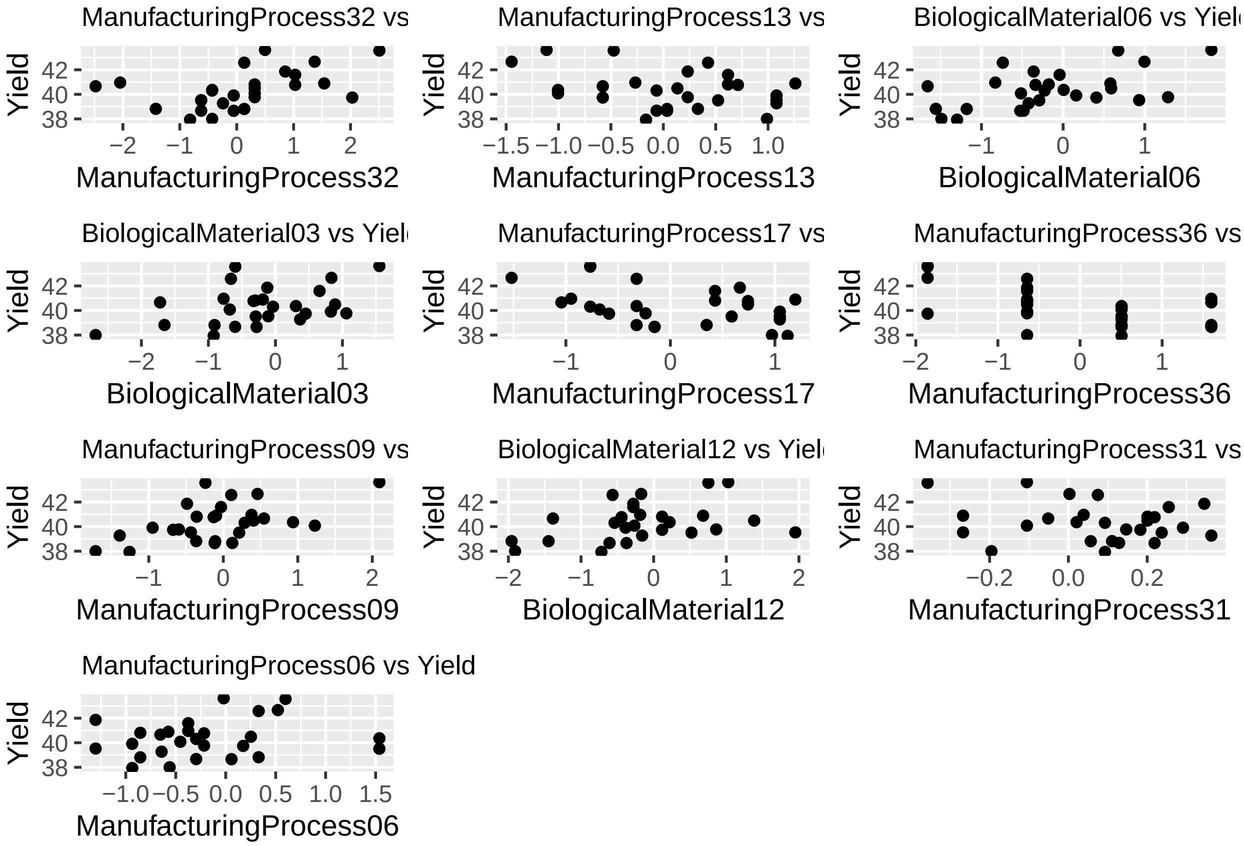
Answer:

A lot of random patterns here but also strong ones. Maufacturing Process 36 has a very specific style of distributiob against yield. I would be curious to know what's driving that. My intuition would now say there will always be some sort of meaningful relationship but it does breakdown along biological/processing lines.

```
top_vars <- varImp(svmtt)$importance %>%
  arrange(desc(Overall)) %>%
  head(10) %>%
  rownames()
besties <- top_vars

plotter <- lapply(besties, function(predictor) {
  ggplot(ctt, aes(x = .data[[predictor]], y = Yield)) +
    geom_point() +
    labs(x = predictor, y = "Yield", title = paste(predictor, "vs Yield")) +
    theme(plot.title = element_text(size = 10))
})

grid.arrange(grobs = plotter)
```


ManufacturingProcess32 vs
Yield
ManufacturingProcess32
ManufacturingProcess13 vs
Yield
ManufacturingProcess13
BiologicalMaterial06 vs Yiel
Yield
BiologicalMaterial06
BiologicalMaterial03 vs Yiel
Yield
BiologicalMaterial03
ManufacturingProcess17 vs
Yield
ManufacturingProcess17
ManufacturingProcess36 vs
Yield
ManufacturingProcess36
ManufacturingProcess09 vs
Yield
ManufacturingProcess09
BiologicalMaterial12 vs Yiel
Yield
BiologicalMaterial12
ManufacturingProcess31 vs
Yield
ManufacturingProcess31
ManufacturingProcess06 vs Yield
Yield
ManufacturingProcess06