KJ Chapter7 Problem 5

Jonathan Hernandez

```
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.

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

(b) Which predictors are most important in the optimal nonlinear regres-
sion 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?

(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?
```

- For (a) let's bring in some of the code from exercise 6.3 then proceed to fit some nonlinear regression models using original resampling (making predictions using the model and a resample of the training set and the test set.)

```
library(AppliedPredictiveModeling)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(e1071)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
data(ChemicalManufacturingProcess)
```

```
# Split up the two components: the response variable as Chem_yield and
# Chem_predictors as the matrix of predictors

Chem_yield <- ChemicalManufacturingProcess$Yield # y vector (response)
Chem_predictors <- ChemicalManufacturingProcess[, 2:58] # X matrix (explanatory vars)
Chem_predictors <- impute(Chem_predictors, what = "median") # apply median to NA values for predictors
```

```
# when evaluating lm model using train() for the first time,
# R complained that there were some
# predictors with zero variance, let's see if there is a way to remove them
zero_var <- nearZeroVar(Chem_predictors)
str(zero_var) # output will show which column has the zero variance
```

```
##  int 7
```

```
# extract features that don't have zero variance in the training set
Chem_predictors <- Chem_predictors[, -zero_var]
```

- Splitting the data into a 70/30 training/test set and doing pre-processing steps

```
set.seed(123) # set seed for reproducible result

# create the data partition 70/30 training/test set
# data_split contains the row indicies to use in the training set, the rest in
# the test set
data_split <- createDataPartition(Chem_yield, p = 0.7)

# Training set X and y
chem_train_X <- Chem_predictors[data_split$Resample1, ]
chem_train_y <- Chem_yield[data_split$Resample1]

# Test set X and y
chem_test_X <- Chem_predictors[-data_split$Resample1, ]
chem_test_y <- Chem_yield[-data_split$Resample1]
```

- Fit a K-NN (Nearest Neighbors) model, make predictions and compute the RMSE for each one and the tuning parameter which in this case is k.

```
knn_chem_model <- train(x=chem_train_X, y=chem_train_y, method = "knn",
                        preProc = c("center", "scale"), tuneLength = 10,
                       metric = "RMSE")
knn_chem_model
```

```
## k-Nearest Neighbors
##
## 124 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 124, 124, 124, 124, 124, 124, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  1.504292  0.3957559  1.193529
##    7  1.505579  0.3867091  1.208774
##    9  1.482170  0.4017305  1.205816
##   11  1.483191  0.3998296  1.205992
##   13  1.482916  0.4066936  1.208522
##   15  1.489299  0.4057495  1.215889
##   17  1.496228  0.4045694  1.217622
##   19  1.493883  0.4158511  1.210827
##   21  1.493323  0.4275953  1.203616
##   23  1.494531  0.4358719  1.204634
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 9.
```

```
# k-NN predictions using the model and the training data after just making a
# k-NN model
chem_knn_pred_training <- predict(knn_chem_model, newdata=as.matrix(chem_train_X))
# standard predictions using the model and the test data
chem_knn_pred_test <- predict(knn_chem_model, newdata=as.matrix(chem_test_X))

# Compute the RMSE between the training responses and predicted training responses
chem_knn_rmse_training <- postResample(pred=chem_knn_pred_training, obs=chem_train_y)
# Compute the RMSE between the test responses and predicted test responses
chem_knn_rmse_test <- postResample(pred=chem_knn_pred_test, obs=chem_test_y)

knn_results <- data.frame(ModelName="k-NN",
                          Training_RMSE=chem_knn_rmse_training[1],
                          Test_RMSE=chem_knn_rmse_test[1])
```

- Fit a Neural Network model and making training and test predictions and finding number of hidden units.

```
set.seed(123)
# Train a neural network model where I specify 10 hidden layers and all predictors
# inputted to each hiden layer (linear combinations of predictors)
neuralnet_chem_model <- train(x=chem_train_X, y=chem_train_y, method = "nnet",
```

```
                         preProc = c("center", "scale"), linout=TRUE,
                         MaxNWts = 10*(ncol(chem_train_X)+1) + 10 + 1,
                         maxit=500, trace=FALSE)
neuralnet_chem_model
```

```
## Neural Network
##
## 124 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 124, 124, 124, 124, 124, 124, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE      Rsquared   MAE
##   1     0e+00  1.827746  0.1536024  1.484291
##   1     1e-04  1.805590  0.2597148  1.432774
##   1     1e-01  2.699736  0.2285453  1.985021
##   3     0e+00  2.422850  0.2037590  1.878368
##   3     1e-04  2.626129  0.2792129  2.020664
##   3     1e-01  3.061575  0.1811179  2.222607
##   5     0e+00  4.216977  0.1166910  3.291888
##   5     1e-04  3.625235  0.1544933  2.737112
##   5     1e-01  2.935044  0.2082395  2.078338
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 1 and decay = 1e-04.
```

```
# Neural Network predictions using the model and the training data after just making a
# Neural Network
chem_neuralnet_pred_training <- predict(neuralnet_chem_model, newdata=as.matrix(chem_train_X))
# standard predictions using the model and the test data
chem_neuralnet_pred_test <- predict(neuralnet_chem_model, newdata=as.matrix(chem_test_X))

# Compute the RMSE between the training responses and predicted training responses
chem_neuralnet_rmse_training <- postResample(pred=chem_neuralnet_pred_training, obs=chem_train_y)
# Compute the RMSE between the test responses and predicted test responses
chem_neuralnet_rmse_test <- postResample(pred=chem_neuralnet_pred_test, obs=chem_test_y)

# results including training/test RMSE
neuralnet_results <- data.frame(ModelName="Neural Network",
                         Training_RMSE=chem_neuralnet_rmse_training[1],
                         Test_RMSE=chem_neuralnet_rmse_test[1])
```

- Fit a MARS (Multivariate Adaptive of Regression Splines) model and making training and test predictions.

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

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

```
## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos
```

```
set.seed(123)
mars_chem_model <- train(x=chem_train_X, y=chem_train_y, method="earth",
                    preProc=c("center", "scale"),
                    tuneGrid = expand.grid(.degree=1:2,.nprune=2:38))
mars_chem_model
```

```
## Multivariate Adaptive Regression Spline
##
## 124 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 124, 124, 124, 124, 124, 124, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1        2      1.695670  0.2744334  1.369920
##   1        3      1.397576  0.5059918  1.131727
##   1        4      1.333796  0.5467072  1.094235
##   1        5      1.771678  0.5435756  1.168514
##   1        6      1.871811  0.5127784  1.197142
##   1        7      2.063034  0.4888595  1.244829
##   1        8      2.030738  0.4847049  1.248494
##   1        9      1.971152  0.4625920  1.243628
##   1       10      2.065662  0.4506927  1.290776
##   1       11      2.245384  0.3987436  1.379900
##   1       12      2.141779  0.4181178  1.356464
##   1       13      2.142350  0.4147741  1.353186
##   1       14      2.374784  0.3694855  1.436574
##   1       15      2.325115  0.3737235  1.421245
##   1       16      2.295667  0.3733943  1.416611
##   1       17      2.326350  0.3702910  1.428126
##   1       18      2.348238  0.3674724  1.437426
##   1       19      2.352639  0.3662749  1.440403
##   1       20      2.367116  0.3619431  1.449243
##   1       21      2.378604  0.3584794  1.454690
##   1       22      2.380095  0.3583756  1.457656
##   1       23      2.379765  0.3586654  1.458084
##   1       24      2.391200  0.3568298  1.463829
##   1       25      2.392479  0.3562738  1.465095
##   1       26      2.392479  0.3562738  1.465095
##   1       27      2.392479  0.3562738  1.465095
##   1       28      2.399125  0.3567644  1.467036
##   1       29      2.404110  0.3564018  1.468383
##   1       30      2.404110  0.3564018  1.468383
##   1       31      2.404110  0.3564018  1.468383
```

```
##   1       32      2.404110  0.3564018  1.468383
##   1       33      2.404110  0.3564018  1.468383
##   1       34      2.404110  0.3564018  1.468383
##   1       35      2.404110  0.3564018  1.468383
##   1       36      2.404110  0.3564018  1.468383
##   1       37      2.404110  0.3564018  1.468383
##   1       38      2.404110  0.3564018  1.468383
##   2        2      1.666222  0.2932730  1.343640
##   2        3      1.420758  0.4794601  1.148270
##   2        4      1.518410  0.5010382  1.150199
##   2        5      1.488081  0.5083620  1.143231
##   2        6      1.628658  0.5010384  1.172405
##   2        7      2.113944  0.4339161  1.295541
##   2        8      2.264870  0.4371688  1.326825
##   2        9      2.364970  0.4261572  1.380621
##   2       10      2.552056  0.4099248  1.426196
##   2       11      2.537859  0.4189566  1.414278
##   2       12      2.992218  0.3808831  1.556614
##   2       13      2.737056  0.3921170  1.488726
##   2       14      2.939048  0.3695614  1.554120
##   2       15      3.417291  0.3460978  1.656564
##   2       16      3.377567  0.3316232  1.668934
##   2       17      3.449211  0.3147242  1.707788
##   2       18      3.478803  0.3195317  1.715817
##   2       19      3.577751  0.3166969  1.759000
##   2       20      3.679034  0.3162148  1.799111
##   2       21      3.691298  0.3116596  1.826735
##   2       22      3.774530  0.3066351  1.848915
##   2       23      3.781006  0.3094914  1.855390
##   2       24      3.824568  0.2918816  1.886362
##   2       25      3.593361  0.2896520  1.857209
##   2       26      3.678474  0.2871199  1.876717
##   2       27      3.710400  0.2859702  1.893867
##   2       28      3.582200  0.2878911  1.877021
##   2       29      3.487636  0.2864917  1.863240
##   2       30      3.494183  0.2856165  1.864619
##   2       31      3.519343  0.2854741  1.873612
##   2       32      3.563908  0.2741150  1.886302
##   2       33      3.559585  0.2754749  1.883397
##   2       34      3.585875  0.2707871  1.891274
##   2       35      3.586332  0.2704944  1.893904
##   2       36      3.586466  0.2709028  1.893392
##   2       37      3.587240  0.2705805  1.894940
##   2       38      3.587240  0.2705805  1.894940
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 4 and degree = 1.
```

```
chem_mars_pred_training <- predict(mars_chem_model, newdata=as.matrix(chem_train_X))
chem_mars_pred_test <- predict(mars_chem_model, newdata=as.matrix(chem_test_X))

chem_mars_rmse_training <- postResample(pred=chem_mars_pred_training, obs=chem_train_y)
chem_mars_rmse_test <- postResample(pred=chem_mars_pred_test, obs=chem_test_y)
```

```
mars_results <- data.frame(ModelName="MARS",
                           Training_RMSE=chem_mars_rmse_training[1],
                           Test_RMSE=chem_mars_rmse_test[1])
```

- Fit a SVM (Support Vector Machine) model using a radial kernel.

```
library(kernlab)
```

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

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

```
set.seed(123)
svm_chem_model <- train(x=chem_train_X, y=chem_train_y, method="svmRadial",
                    preProc=c("center", "scale"), tuneLength = 20)

svm_chem_model
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 124 samples
##  56 predictor
##
## Pre-processing: centered (56), scaled (56)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 124, 124, 124, 124, 124, 124, ...
## Resampling results across tuning parameters:
##
##   C          RMSE      Rsquared   MAE
##        0.25  1.526548  0.4705106  1.233186
##        0.50  1.426807  0.5118907  1.151948
##        1.00  1.344031  0.5497810  1.087846
##        2.00  1.288030  0.5794707  1.042786
##        4.00  1.266904  0.5899731  1.021170
##        8.00  1.257491  0.5943592  1.012916
##       16.00  1.256824  0.5946499  1.012214
##       32.00  1.256824  0.5946499  1.012214
##       64.00  1.256824  0.5946499  1.012214
##      128.00  1.256824  0.5946499  1.012214
##      256.00  1.256824  0.5946499  1.012214
##      512.00  1.256824  0.5946499  1.012214
##     1024.00  1.256824  0.5946499  1.012214
##     2048.00  1.256824  0.5946499  1.012214
##     4096.00  1.256824  0.5946499  1.012214
##     8192.00  1.256824  0.5946499  1.012214
##    16384.00  1.256824  0.5946499  1.012214
##    32768.00  1.256824  0.5946499  1.012214
##    65536.00  1.256824  0.5946499  1.012214
```

```
##   131072.00  1.256824  0.5946499  1.012214
##
## Tuning parameter 'sigma' was held constant at a value of 0.0134794
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.0134794 and C = 16.
```

```
chem_svm_pred_training <- predict(svm_chem_model, newdata=as.matrix(chem_train_X))
chem_svm_pred_test <- predict(svm_chem_model, newdata=as.matrix(chem_test_X))

chem_svm_rmse_training <- postResample(pred=chem_svm_pred_training, obs=chem_train_y)
chem_svm_rmse_test <- postResample(pred=chem_svm_pred_test, obs=chem_test_y)

svm_results <- data.frame(ModelName="SVM",
                          Training_RMSE=chem_svm_rmse_training[1],
                          Test_RMSE=chem_svm_rmse_test[1])
```

- Let's look at the results of each nonlinear model and see how the RMSE evaluates in testing and in training data:

```
# summary of metrics
model_results <- do.call("rbind", list(knn_results,
                                       neuralnet_results,
                                       mars_results,
                                       svm_results))
rownames(model_results) <- NULL
model_results
```

```
##        ModelName Training_RMSE Test_RMSE
## 1           k-NN     1.2591479  1.104242
## 2 Neural Network     1.2086264  1.572906
## 3           MARS     1.1874477  1.195969
## 4            SVM     0.1889792  1.055539
```

- What we see is that the MARS model has the lowest training RMSE while the Neural Network has the smallest test RMSE. Also, the MARS model has a slightly higher RMSE than the Neural Network
- For (b) the Neural Network model will be used and we extract the features of the model

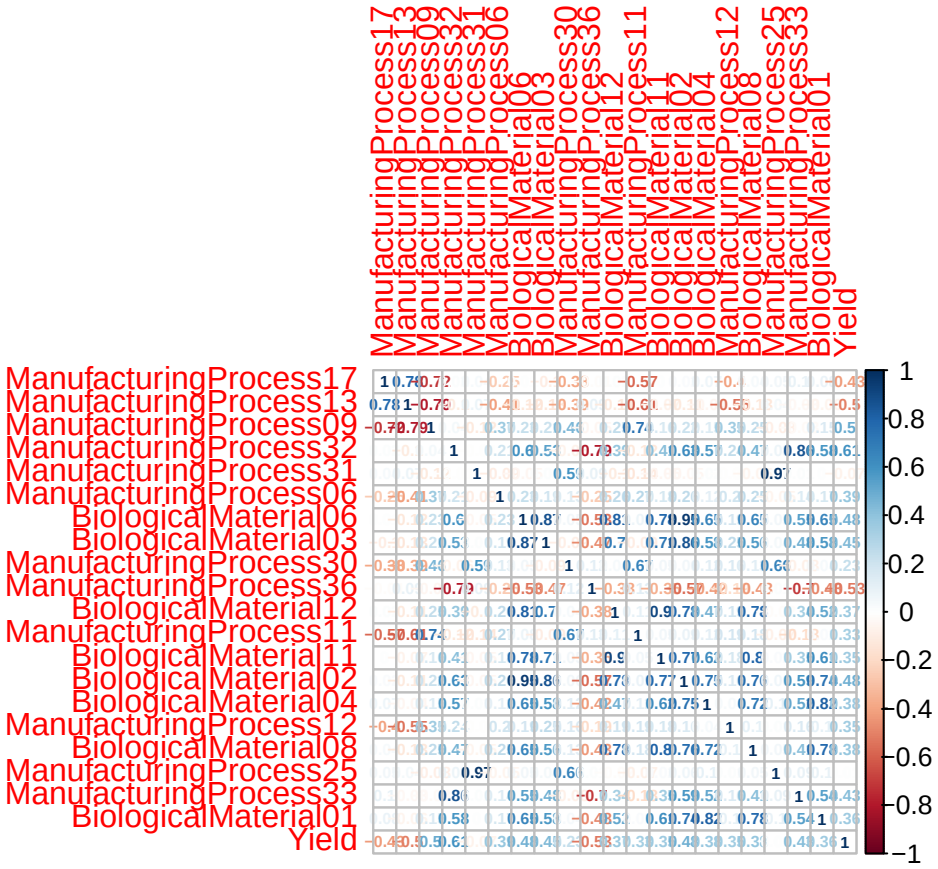
```
important_nnet_vars <- varImp(svm_chem_model)
important_nnet_vars
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 56)
##
##                         Overall
## ManufacturingProcess17  100.00
## ManufacturingProcess13   98.14
## ManufacturingProcess09   77.57
## ManufacturingProcess32   72.91
## ManufacturingProcess31   66.26
## ManufacturingProcess06   64.56
```

```
## BiologicalMaterial06     59.81
## BiologicalMaterial03     57.30
## ManufacturingProcess30   55.20
## ManufacturingProcess36   54.24
## BiologicalMaterial12     52.90
## ManufacturingProcess11   52.22
## BiologicalMaterial11     46.44
## BiologicalMaterial02     45.91
## BiologicalMaterial04     42.33
## ManufacturingProcess12   37.40
## BiologicalMaterial08     37.32
## ManufacturingProcess25   35.79
## ManufacturingProcess33   31.28
## BiologicalMaterial01     30.49
```

- We see that of the 20 most important features, 4 manufactring process features make the top 4 with BiologicalMaterial05 coming in 5th place and BiologicalMaterial01 coming in 16th place. The rest of the variables are mostly ManufacturingProcess features and therefore are the most important in the model.
- How do the top 10 predictor variables in our nonlinear model compare to the linear one from 6.3?
- The 'BiologicalMaterial05' is in the top 10 for both models.
- 'ManufacturingProcess36' is the most important predictor in both models
- Variables "ManufacturingProcess36", "ManufacturingProcess37", "ManufacturingProcess32", "BiologicalMaterial05" appear in both models and the rest of the top 10 for each one are in either one model or the other.
- For (c), let's do a correlation plot of the features from the neural network model

```
# convert to data frame to extract column names of top features of importance
important_nnet_vars.df <- data.frame(Features=rownames(important_nnet_vars$importance),
                                     Imp=important_nnet_vars$importance$Overall)
# sort by overall importance
important_nnet_vars.df <- important_nnet_vars.df[order(-important_nnet_vars.df$Imp),]
# convert to character the features so they can be properly selected in the original
# dataset
features <- as.character(important_nnet_vars.df[1:20, "Features"])
corrplot(cor(ChemicalManufacturingProcess[c(features,"Yield")],
    use = "pairwise.complete.obs"),
    tl.cex = 1, method = "number",
    number.cex = 0.5)
```



- Much like the Correlation Plot with the lasso model, we see that the highest correlation value with Yield is the ManufacturingProcess32 feature.