Data 624: Week 9 Homework

Angrand, Burke, Deboch, Groysman, Karr

December 8, 2019

##### Chapter 7 KJ 7.2, 7.5

## Exercise 7.2

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

where the values are random variables uniformly distributed between [0,1] (there are also 5 other non-informative variables created in the simulation). The package *mlbench* contains a function called mlbench.friedman1 that simulates these data:

### Read Data & EDA

**a. Creating Training and Testing Data**

set.seed(100)  
trainingData <- mlbench.friedman1(200, sd = 1)  
## We convert th 'x' data from a matrix to data frame  
## One reason is that this will give the columns names.  
trainingData$x <- data.frame(trainingData$x)  
  
  
  
## This creates a list with a vector 'y' and a matrix  
## of predictors 'x'. Also simulate a large test set to  
## estimate the truee error rate with good precisions:  
testData <- mlbench.friedman1(5000, sd = 1)  
testData$x <- data.frame(testData$x)

**The relationship between the predictors x1-x10 and the response y**

head(trainingData$x)

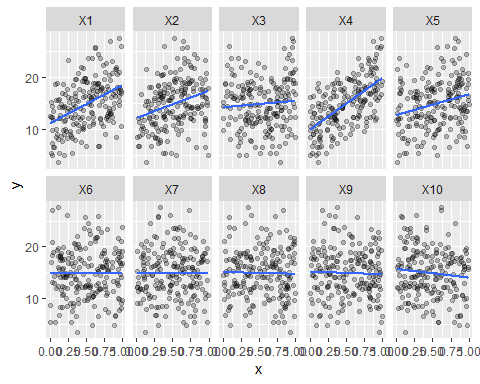
## X1 X2 X3 X4 X5 X6 X7  
## 1 0.30776611 0.3695961 0.5112374 0.03176634 0.09942609 0.0740483 0.9734581  
## 2 0.25767250 0.9563228 0.2777107 0.57970549 0.22993408 0.1118664 0.7717425  
## 3 0.55232243 0.9135767 0.3606569 0.15420484 0.44362621 0.6239440 0.4949800  
## 4 0.05638315 0.8233363 0.4375279 0.12527050 0.51570490 0.6710818 0.3926061  
## 5 0.46854928 0.3194822 0.8030667 0.14798581 0.92489425 0.3658942 0.1053059  
## 6 0.48377074 0.8777003 0.5206097 0.91334263 0.04445684 0.1831814 0.3560706  
## X8 X9 X10  
## 1 0.6939725 0.3761842 0.5704419  
## 2 0.2022793 0.2294918 0.4786617  
## 3 0.5222016 0.9693079 0.3072272  
## 4 0.8133963 0.1315974 0.8855426  
## 5 0.9161582 0.6090285 0.5418244  
## 6 0.9463884 0.2840458 0.4221332

head(testData$x)

## X1 X2 X3 X4 X5 X6  
## 1 0.9511717 0.84353248 0.8396137 0.3112072 0.02490033 0.42544544  
## 2 0.9223041 0.58380180 0.5400547 0.6214827 0.29711901 0.02439103  
## 3 0.3012870 0.85267526 0.3482427 0.8339510 0.60000986 0.79394431  
## 4 0.5742679 0.42253889 0.9993041 0.2476972 0.96193464 0.99934132  
## 5 0.7283813 0.67991180 0.7148379 0.7698435 0.68527210 0.38128356  
## 6 0.5430245 0.07116613 0.4131925 0.7132922 0.01547487 0.93686149  
## X7 X8 X9 X10  
## 1 0.27018771 0.9954754 0.7930478 0.53310722  
## 2 0.53027543 0.3503692 0.9809181 0.67506119  
## 3 0.03388816 0.4602326 0.2053274 0.58461319  
## 4 0.01742161 0.1520394 0.8638043 0.06242131  
## 5 0.69713633 0.8956982 0.6678364 0.09462883  
## 6 0.89782147 0.5493450 0.7149965 0.72507148

**b. Determine correlation between x and y**

## Look at the data using featurePlot  
## or other methods.  
  
trainingData$x %>%  
 #gather x and y  
 mutate(y=trainingData$y) %>%  
 # tidy data frame for easier manipulating & plotting  
 gather(var, x,-y) %>%  
 #factor x variable and change factors so X10 is last  
 mutate(var= forcats::fct\_relevel(factor(var), "X10", after=Inf)) %>%  
 ggplot(aes(x,y)) +  
 geom\_point(alpha=0.25) +  
 stat\_smooth(method="glm", se = FALSE) +  
 facet\_wrap(~ var, nrow = 2)



### Training Models

Tune several models on these data. For example:

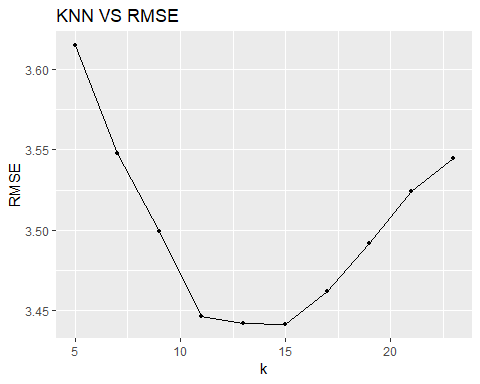
**a. K-Nearest Neighboor Model (KNN)**

library(caret)  
set.seed(921)  
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 3.614818 0.4443500 2.959250  
## 7 3.547913 0.4721382 2.918330  
## 9 3.499002 0.4998668 2.879042  
## 11 3.446309 0.5310551 2.825429  
## 13 3.441987 0.5462941 2.822529  
## 15 3.441374 0.5645635 2.815643  
## 17 3.462089 0.5718349 2.824125  
## 19 3.491635 0.5728072 2.843148  
## 21 3.524142 0.5697454 2.879433  
## 23 3.544471 0.5755214 2.894378  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 15.

RMSE with varying tuning parameters

knnModel$results %>%  
 ggplot(aes(x=k, y=RMSE)) +  
 geom\_line() + geom\_point(size=1) +  
 labs(title="KNN VS RMSE")



* k=15 is the optimal model

knnPred <- predict(knnModel, newdata = testData$x)  
  
## The function 'postResample' can be used to get test set  
## performance values  
knn.pred <- postResample(pred = knnPred, obs = testData$y)  
knn.pred

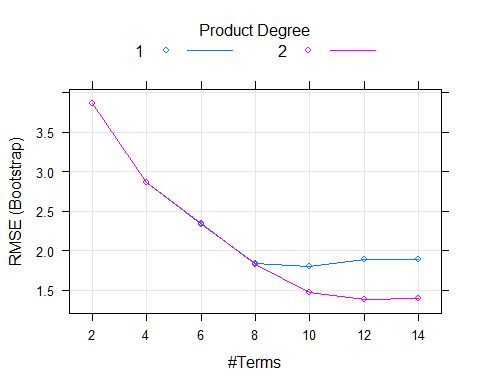
## RMSE Rsquared MAE   
## 3.3709432 0.6630201 2.7279373

Which model appears to get the best performance? Does MARS select the informative predictors (those named X1-15)

K-nearest neighbors models perform better when predictor and response relationships have a locational dependency. The simulation data is not related in this way so other models are expected perform better. In fact MARS and SVM have lower RMSE values and thus a better fit.

**b. MARS Model**

marsGrid <- expand.grid(degree = 1:2, nprune = seq(2,14,by=2))  
set.seed(921)  
marsModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "earth",  
 preProc = c("center","scale"),  
 tuneGrid = marsGrid)  
  
marsPred <- predict(marsModel, newdata = testData$x)  
plot(marsModel)



mars.pred <- postResample(pred = marsPred, obs = testData$y)  
mars.pred

## RMSE Rsquared MAE   
## 1.1772309 0.9430908 0.9386423

* The MARS model is the optimal one of those tested with the lowest RMSE or fit. The optimal RMSE is achieved with a second-degree. We can further investigate variable importance and see that only the top 5 predictors have significant influence on the response variable with the following ranking . . . V4, V1, V3, V5, V3.

varImp(marsModel)

## earth variable importance  
##   
## Overall  
## X4 100.00  
## X1 78.53  
## X2 67.42  
## X5 52.24  
## X3 40.90  
## X7 0.00  
## X10 0.00  
## X8 0.00  
## X9 0.00  
## X6 0.00

* A summary model can also be generated using the earth function

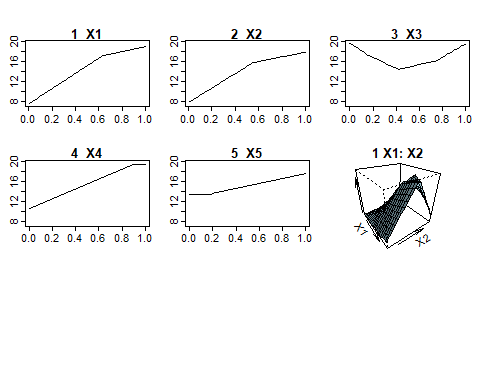
marsFit <- earth(x = trainingData$x,  
 y = trainingData$y,  
 nprune = 12, degree = 2)  
summary(marsFit)

## Call: earth(x=trainingData$x, y=trainingData$y, degree=2, nprune=12)  
##   
## coefficients  
## (Intercept) 18.273216  
## h(0.629995-X1) -16.509679  
## h(X1-0.629995) 4.989265  
## h(0.55336-X2) -19.474961  
## h(X2-0.55336) 6.462382  
## h(X3-0.162376) 4.970899  
## h(0.422184-X3) 15.463906  
## h(X3-0.741199) 8.489787  
## h(0.902811-X4) -9.745742  
## h(X5-0.161512) 5.006716  
## h(X1-0.468549) \* h(X2-0.55336) -55.417177  
## h(0.673249-X1) \* h(0.55336-X2) 28.647676  
##   
## Selected 12 of 19 terms, and 5 of 10 predictors  
## Termination condition: Reached nk 21  
## Importance: X4, X1, X2, X5, X3, X6-unused, X7-unused, X8-unused, ...  
## Number of terms at each degree of interaction: 1 9 2  
## GCV 1.381629 RSS 203.1841 GRSq 0.940204 RSq 0.9555886

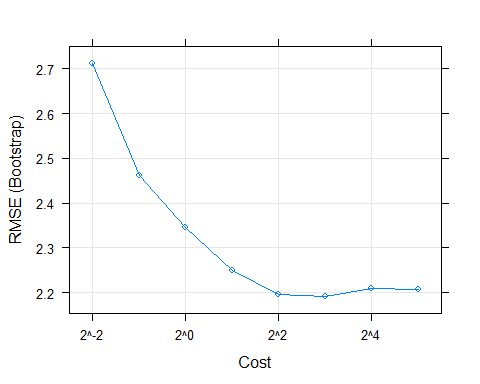
**c. SVM Model**

plotmo(marsFit, caption = "")

## plotmo grid: X1 X2 X3 X4 X5 X6  
## 0.5011993 0.5111177 0.5632154 0.4727849 0.5352105 0.5118565  
## X7 X8 X9 X10  
## 0.5070687 0.5548462 0.509134 0.4232667



set.seed(921)  
svmRModel <- train(x = trainingData$x,   
 y = trainingData$y,  
 method = "svmRadial",  
 preProc = c("center","scale"),  
 tuneLength = 8)  
svmRPred <- predict(svmRModel, newdata = testData$x)  
  
svm.pred <- postResample(pred = svmRPred, obs = testData$y)  
plot(svmRModel, scales = list(x = list(log = 2)))



* The Cost to RMSE(Bootstrap) plot shows the SVM tuning parameter profile. The optimal model has a cost value of 16 and an RMSE of ~2.0%

rbind(knn.pred, mars.pred, svm.pred)

## RMSE Rsquared MAE  
## knn.pred 3.370943 0.6630201 2.7279373  
## mars.pred 1.177231 0.9430908 0.9386423  
## svm.pred 1.962450 0.8418429 1.5185282

* Overall, the MARS model performs best, the radial basis function SVM coming in next and K-NN has the worst performance for this problem.

## Exercise 7.5

7.5 Exercise 6.3 describes data for a chemical manufacturing process. Use the smae data imputation, data splitting and pre-processing steps as before and train several nonlinear regressions models.  
  
a) Which nonlinear regression model gives the optimal resampling and test set performance?  
  
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?  
   
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 bioglogical or process predictors and their relationship yield?

### Data Pre-Work

**a. Read in Data & EDA**

set.seed(100)  
data(ChemicalManufacturingProcess)  
  
processPredictors = ChemicalManufacturingProcess[,2:58]  
yield = ChemicalManufacturingProcess[,1]  
  
n\_samples = dim(processPredictors)[1]  
n\_features = dim(processPredictors)[2]  
n\_samples

## [1] 176

n\_features

## [1] 57

**b. Impute missing values**

null.values <-as.data.frame(sapply(processPredictors, function(x) sum(is.na(x))), col.names = "null\_values")%>%  
 tibble::rownames\_to\_column("Predictors")%>%  
 rename(nulls = 2)%>%  
 arrange(-nulls)%>%  
 filter(nulls >0)  
null.values

## Predictors nulls  
## 1 ManufacturingProcess03 15  
## 2 ManufacturingProcess11 10  
## 3 ManufacturingProcess10 9  
## 4 ManufacturingProcess25 5  
## 5 ManufacturingProcess26 5  
## 6 ManufacturingProcess27 5  
## 7 ManufacturingProcess28 5  
## 8 ManufacturingProcess29 5  
## 9 ManufacturingProcess30 5  
## 10 ManufacturingProcess31 5  
## 11 ManufacturingProcess33 5  
## 12 ManufacturingProcess34 5  
## 13 ManufacturingProcess35 5  
## 14 ManufacturingProcess36 5  
## 15 ManufacturingProcess02 3  
## 16 ManufacturingProcess06 2  
## 17 ManufacturingProcess01 1  
## 18 ManufacturingProcess04 1  
## 19 ManufacturingProcess05 1  
## 20 ManufacturingProcess07 1  
## 21 ManufacturingProcess08 1  
## 22 ManufacturingProcess12 1  
## 23 ManufacturingProcess14 1  
## 24 ManufacturingProcess22 1  
## 25 ManufacturingProcess23 1  
## 26 ManufacturingProcess24 1  
## 27 ManufacturingProcess40 1  
## 28 ManufacturingProcess41 1

# Fill in missing values where we have NAs with the median over the non-NA values:   
replacements = sapply( processPredictors, median, na.rm=TRUE )  
as.data.frame(replacements)

## replacements  
## BiologicalMaterial01 6.305  
## BiologicalMaterial02 55.090  
## BiologicalMaterial03 67.220  
## BiologicalMaterial04 12.100  
## BiologicalMaterial05 18.490  
## BiologicalMaterial06 48.460  
## BiologicalMaterial07 100.000  
## BiologicalMaterial08 17.510  
## BiologicalMaterial09 12.835  
## BiologicalMaterial10 2.710  
## BiologicalMaterial11 146.080  
## BiologicalMaterial12 20.120  
## ManufacturingProcess01 11.400  
## ManufacturingProcess02 21.000  
## ManufacturingProcess03 1.540  
## ManufacturingProcess04 934.000  
## ManufacturingProcess05 999.200  
## ManufacturingProcess06 206.800  
## ManufacturingProcess07 177.000  
## ManufacturingProcess08 178.000  
## ManufacturingProcess09 45.730  
## ManufacturingProcess10 9.100  
## ManufacturingProcess11 9.400  
## ManufacturingProcess12 0.000  
## ManufacturingProcess13 34.600  
## ManufacturingProcess14 4856.000  
## ManufacturingProcess15 6031.500  
## ManufacturingProcess16 4588.000  
## ManufacturingProcess17 34.400  
## ManufacturingProcess18 4835.000  
## ManufacturingProcess19 6022.000  
## ManufacturingProcess20 4582.000  
## ManufacturingProcess21 -0.300  
## ManufacturingProcess22 5.000  
## ManufacturingProcess23 3.000  
## ManufacturingProcess24 8.000  
## ManufacturingProcess25 4855.000  
## ManufacturingProcess26 6047.000  
## ManufacturingProcess27 4587.000  
## ManufacturingProcess28 10.400  
## ManufacturingProcess29 19.900  
## ManufacturingProcess30 9.100  
## ManufacturingProcess31 70.800  
## ManufacturingProcess32 158.000  
## ManufacturingProcess33 64.000  
## ManufacturingProcess34 2.500  
## ManufacturingProcess35 495.000  
## ManufacturingProcess36 0.020  
## ManufacturingProcess37 1.000  
## ManufacturingProcess38 3.000  
## ManufacturingProcess39 7.200  
## ManufacturingProcess40 0.000  
## ManufacturingProcess41 0.000  
## ManufacturingProcess42 11.600  
## ManufacturingProcess43 0.800  
## ManufacturingProcess44 1.900  
## ManufacturingProcess45 2.200

for( ci in 1:n\_features ){  
 bad\_inds = is.na( processPredictors[,ci] )  
 processPredictors[bad\_inds,ci] = replacements[ci]  
}

**c. No Variance Predictors Removal**

# Look for any features with no variance:  
zero\_cols = nearZeroVar( processPredictors )  
zero\_cols

## [1] 7

processPredictors = processPredictors[,-zero\_cols] # drop these zero variance columns

**d. Train/Test Split**

# Split this data into training and testing sets:  
# We set aside 20% of the observations to be the test dataset.  
training = createDataPartition( yield, p=0.8 )  
  
processPredictors\_training = processPredictors[training$Resample1,]  
yield\_training = yield[training$Resample1]  
  
processPredictors\_testing = processPredictors[-training$Resample1,]  
yield\_testing = yield[-training$Resample1]  
preProc\_Arguments = c("center","scale")

### Data Modeling

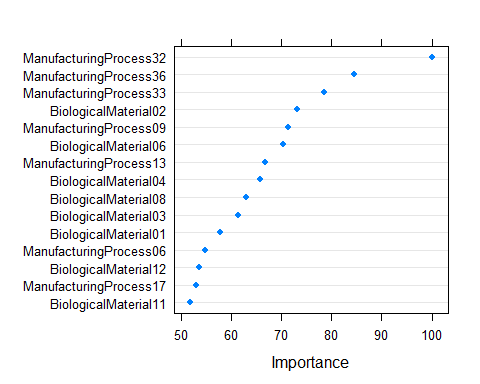
**a. PLS**

* adding thepls for comp. purposes

set.seed(100)  
plsModel<-train(x=processPredictors\_training, y=yield\_training, method="pls", tuneLength = 10,preProcess=preProc\_Arguments)  
plsModel

## Partial Least Squares   
##   
## 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:  
##   
## ncomp RMSE Rsquared MAE   
## 1 1.606147 0.3348312 1.192398  
## 2 2.616416 0.2525362 1.344721  
## 3 2.261047 0.3026285 1.276079  
## 4 2.249021 0.3164123 1.281933  
## 5 2.543660 0.2914832 1.354735  
## 6 2.838645 0.2619764 1.432249  
## 7 3.249180 0.2450920 1.530636  
## 8 3.534123 0.2352648 1.606542  
## 9 3.788171 0.2258456 1.668000  
## 10 4.015881 0.2112639 1.722710  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was ncomp = 1.

# Lets see what variables are most important in the pls model:   
dotPlot(varImp(plsModel), top=15)



**b. KNN**

# A K-NN model:  
set.seed(100)  
knnModel = train(x=processPredictors\_training, y=yield\_training, method="knn", preProc=preProc\_Arguments, tuneLength=10)  
  
# predict on training/testing sets  
knnPred = predict(knnModel, newdata=processPredictors\_training)  
knnPR = postResample(pred=knnPred, obs=yield\_training)  
rmses\_training = c(knnPR[1])  
r2s\_training = c(knnPR[2])  
methods = c("KNN")  
  
pred.train.knn<- data.frame(cbind(rmses\_training, r2s\_training))  
  
knnPred = predict(knnModel, newdata=processPredictors\_testing)  
knnPR = postResample(pred=knnPred, obs=yield\_testing)  
rmses\_testing = c(knnPR[1])  
r2s\_testing = c(knnPR[2])  
  
pred.test.knn<- data.frame(cbind(rmses\_testing, r2s\_testing))  
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 1.470315 0.3687432 1.145282  
## 7 1.441013 0.3916406 1.131379  
## 9 1.439163 0.3954489 1.136648  
## 11 1.436249 0.4026296 1.141582  
## 13 1.434620 0.4080690 1.141594  
## 15 1.438548 0.4121259 1.143584  
## 17 1.444909 0.4126914 1.149710  
## 19 1.450598 0.4140712 1.149589  
## 21 1.460501 0.4116571 1.157581  
## 23 1.472091 0.4033518 1.166578  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 13.

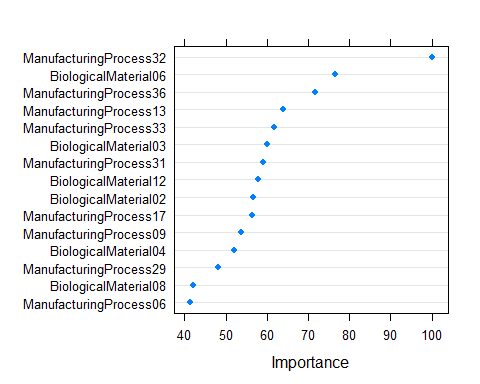
pred.train.knn

## rmses\_training r2s\_training  
## RMSE 1.241575 0.566462

pred.test.knn

## rmses\_testing r2s\_testing  
## RMSE 1.596003 0.4502603

# Lets see what variables are most important in the MARS model:   
dotPlot(varImp(knnModel), top=15)



**c. MARS**

# MARS model:  
marsGrid = expand.grid(.degree=1:2, .nprune=2:38)  
set.seed(100)  
marsModel = train(x=processPredictors\_training, y=yield\_training, method="earth", preProc=preProc\_Arguments, tuneGrid=marsGrid)  
   
marsPred = predict(marsModel, newdata=processPredictors\_training)  
marsPR = postResample(pred=marsPred, obs=yield\_training)  
rmses\_training = c(rmses\_training,marsPR[1])  
r2s\_training = c(r2s\_training,marsPR[2])  
methods = c(methods,"MARS")  
  
pred.train.mars<- data.frame(cbind(rmses\_training, r2s\_training))  
  
marsPred = predict(marsModel, newdata=processPredictors\_testing)  
marsPR = postResample(pred=marsPred, obs=yield\_testing)  
rmses\_testing = c(rmses\_testing,marsPR[1])  
r2s\_testing = c(r2s\_testing,marsPR[2])  
  
pred.test.mars<- data.frame(cbind(rmses\_testing, r2s\_testing))  
  
  
marsModel

## Multivariate Adaptive Regression Spline   
##   
## 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:  
##   
## degree nprune RMSE Rsquared MAE   
## 1 2 1.337151 0.4790123 1.044371  
## 1 3 2.451967 0.5246877 1.163454  
## 1 4 2.659078 0.4983851 1.202188  
## 1 5 2.427845 0.4858812 1.186132  
## 1 6 2.844089 0.4433150 1.266668  
## 1 7 3.310084 0.4204201 1.351856  
## 1 8 2.700904 0.4379416 1.269914  
## 1 9 3.495957 0.3916653 1.402524  
## 1 10 3.792364 0.3805513 1.446673  
## 1 11 4.177186 0.3593005 1.516468  
## 1 12 3.775477 0.3688420 1.478919  
## 1 13 4.680886 0.3275937 1.618349  
## 1 14 4.383436 0.3115230 1.594634  
## 1 15 4.582657 0.3090088 1.626380  
## 1 16 4.236303 0.3081437 1.588525  
## 1 17 4.302603 0.3088288 1.597112  
## 1 18 4.286381 0.3097487 1.595393  
## 1 19 4.289130 0.3094598 1.596080  
## 1 20 4.291710 0.3093852 1.597559  
## 1 21 4.296745 0.3087382 1.598127  
## 1 22 4.296745 0.3087382 1.598127  
## 1 23 4.302942 0.3082131 1.603612  
## 1 24 4.305441 0.3077901 1.604965  
## 1 25 4.304930 0.3071020 1.605520  
## 1 26 4.305669 0.3075048 1.605704  
## 1 27 4.307095 0.3069246 1.606610  
## 1 28 4.307332 0.3066939 1.608166  
## 1 29 4.309387 0.3063270 1.608482  
## 1 30 4.315124 0.3057524 1.612510  
## 1 31 4.311237 0.3053641 1.611400  
## 1 32 4.309146 0.3046149 1.609631  
## 1 33 4.310194 0.3037503 1.608994  
## 1 34 4.311315 0.3048814 1.610440  
## 1 35 4.307734 0.3048285 1.608950  
## 1 36 4.330227 0.3007319 1.614174  
## 1 37 4.334214 0.2998526 1.613614  
## 1 38 4.322701 0.3010904 1.610889  
## 2 2 1.345555 0.4714679 1.053690  
## 2 3 2.471280 0.5081173 1.179458  
## 2 4 2.598490 0.4778338 1.209053  
## 2 5 2.696246 0.4326667 1.259740  
## 2 6 2.697818 0.4231996 1.271407  
## 2 7 3.113208 0.3916284 1.340831  
## 2 8 3.067344 0.3824652 1.332873  
## 2 9 3.164055 0.3850711 1.358673  
## 2 10 3.013927 0.3840193 1.342080  
## 2 11 3.865343 0.3364183 1.581944  
## 2 12 4.166328 0.3073726 1.631589  
## 2 13 4.160213 0.2927092 1.572803  
## 2 14 4.674119 0.2697273 1.755725  
## 2 15 4.774054 0.2683552 1.766710  
## 2 16 4.870083 0.2608880 1.793716  
## 2 17 4.688700 0.2343572 1.778118  
## 2 18 5.180577 0.2367059 1.861901  
## 2 19 5.421072 0.2232073 1.910054  
## 2 20 5.309574 0.1980600 1.911964  
## 2 21 5.792738 0.1861919 2.000826  
## 2 22 5.899997 0.1830459 2.076372  
## 2 23 5.915569 0.1812768 2.106135  
## 2 24 6.542384 0.1742493 2.232929  
## 2 25 6.620992 0.1711038 2.261212  
## 2 26 6.605927 0.1700223 2.260263  
## 2 27 6.739054 0.1719858 2.279209  
## 2 28 6.962719 0.1643088 2.334231  
## 2 29 6.948969 0.1620855 2.336345  
## 2 30 6.881643 0.1664310 2.339031  
## 2 31 7.028239 0.1667145 2.364606  
## 2 32 7.083725 0.1643350 2.387525  
## 2 33 7.100343 0.1636304 2.390179  
## 2 34 7.019171 0.1631205 2.377913  
## 2 35 7.238098 0.1623559 2.434382  
## 2 36 7.242257 0.1605259 2.437154  
## 2 37 7.247018 0.1610770 2.440052  
## 2 38 7.300230 0.1609540 2.455346  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were nprune = 2 and degree = 1.

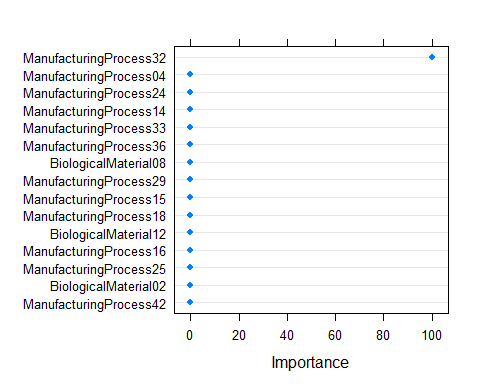
pred.train.mars

## rmses\_training r2s\_training  
## RMSE 1.241575 0.5664620  
## RMSE.1 1.301557 0.4705536

pred.test.mars

## rmses\_testing r2s\_testing  
## RMSE 1.596003 0.4502603  
## RMSE.1 1.823847 0.2278359

# Lets see what variables are most important in the MARS model:   
dotPlot(varImp(marsModel), top=15)



**d. SVM**

# A Support Vector Machine (SVM):  
set.seed(100)  
svmModel = train(x=processPredictors\_training, y=yield\_training, method="svmRadial", preProc=preProc\_Arguments, tuneLength=20)  
  
svmPred = predict(svmModel, newdata=processPredictors\_training)  
svmPR = postResample(pred=svmPred, obs=yield\_training)   
rmses\_training = c(rmses\_training,svmPR[1])  
r2s\_training = c(r2s\_training,svmPR[2])  
methods = c(methods,"SVM")  
  
pred.train.svm<- data.frame(cbind(rmses\_training, r2s\_training))  
  
svmPred = predict(svmModel, newdata=processPredictors\_testing)  
svmPR = postResample(pred=svmPred, obs=yield\_testing)  
rmses\_testing = c(rmses\_testing,svmPR[1])  
r2s\_testing = c(r2s\_testing,svmPR[2])  
  
  
  
pred.test.svm<- data.frame(cbind(rmses\_testing, r2s\_testing))  
  
svmModel

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 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:  
##   
## C RMSE Rsquared MAE   
## 0.25 1.464937 0.4214672 1.158680  
## 0.50 1.390638 0.4539078 1.094862  
## 1.00 1.344224 0.4783981 1.059965  
## 2.00 1.312820 0.4974055 1.035729  
## 4.00 1.295658 0.5088374 1.018830  
## 8.00 1.290419 0.5125148 1.012466  
## 16.00 1.290462 0.5124509 1.012496  
## 32.00 1.290462 0.5124509 1.012496  
## 64.00 1.290462 0.5124509 1.012496  
## 128.00 1.290462 0.5124509 1.012496  
## 256.00 1.290462 0.5124509 1.012496  
## 512.00 1.290462 0.5124509 1.012496  
## 1024.00 1.290462 0.5124509 1.012496  
## 2048.00 1.290462 0.5124509 1.012496  
## 4096.00 1.290462 0.5124509 1.012496  
## 8192.00 1.290462 0.5124509 1.012496  
## 16384.00 1.290462 0.5124509 1.012496  
## 32768.00 1.290462 0.5124509 1.012496  
## 65536.00 1.290462 0.5124509 1.012496  
## 131072.00 1.290462 0.5124509 1.012496  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.01491264  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were sigma = 0.01491264 and C = 8.

pred.train.svm

## rmses\_training r2s\_training  
## RMSE 1.2415750 0.5664620  
## RMSE.1 1.3015567 0.4705536  
## RMSE.2 0.1773248 0.9924706

pred.test.svm

## rmses\_testing r2s\_testing  
## RMSE 1.596003 0.4502603  
## RMSE.1 1.823847 0.2278359  
## RMSE.2 1.216252 0.6565869

### Questions - Answered

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

* The test data used for predictions for KNN, MARS and SVM had RMSE values of 1.59, 1.82 and 1.21 respectively. The SVM model achieved this fit and appears to be the optimal model of those attempted.

# Package the results up:  
res\_training = data.frame( rmse=rmses\_training, r2=r2s\_training )  
rownames(res\_training) <- methods  
  
training\_order = order( -res\_training$rmse )  
  
res\_training = res\_training[ training\_order, ] # Order the dataframe so that the best results are at the bottom:  
print("Final Training Results")

## [1] "Final Training Results"

res\_training

## rmse r2  
## MARS 1.3015567 0.4705536  
## KNN 1.2415750 0.5664620  
## SVM 0.1773248 0.9924706

res\_testing = data.frame( rmse=rmses\_testing, r2=r2s\_testing )  
rownames(res\_testing) = methods  
  
res\_testing = res\_testing[ training\_order, ] # Order the dataframe so that the best results for the training set are at the bottom:  
print("Final Testing Results")

## [1] "Final Testing Results"

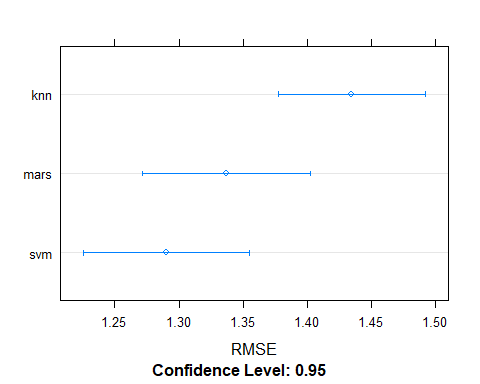
res\_testing

## rmse r2  
## MARS 1.823847 0.2278359  
## KNN 1.596003 0.4502603  
## SVM 1.216252 0.6565869

resamp = resamples( list(knn=knnModel,svm=svmModel,mars=marsModel) )  
summary(resamp)

##   
## Call:  
## summary.resamples(object = resamp)  
##   
## Models: knn, svm, mars   
## Number of resamples: 25   
##   
## MAE   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## knn 0.9789189 1.0674709 1.1443701 1.141594 1.215663 1.365669 0  
## svm 0.8362991 0.9339826 0.9990398 1.012466 1.091015 1.278862 0  
## mars 0.8135195 0.9580927 1.0438588 1.044371 1.124134 1.303500 0  
##   
## RMSE   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## knn 1.1488511 1.311098 1.439503 1.434620 1.554134 1.644363 0  
## svm 1.0351677 1.168407 1.234980 1.290419 1.424835 1.603327 0  
## mars 0.9904353 1.228108 1.345768 1.337151 1.453645 1.599669 0  
##   
## Rsquared   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## knn 0.2375618 0.3604480 0.4134720 0.4080690 0.4597024 0.5372964 0  
## svm 0.2900938 0.4737095 0.5370702 0.5125148 0.5813574 0.6268504 0  
## mars 0.2710663 0.4423782 0.4777335 0.4790123 0.5385581 0.6461168 0

dotplot( resamp, metric="RMSE" )

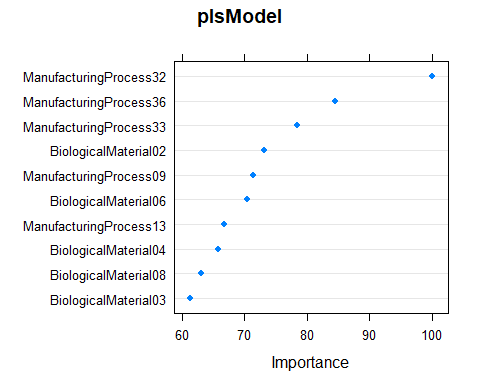


summary(diff(resamp))

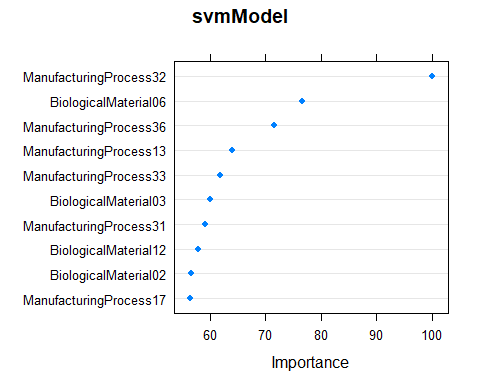
##   
## Call:  
## summary.diff.resamples(object = diff(resamp))  
##   
## p-value adjustment: bonferroni   
## Upper diagonal: estimates of the difference  
## Lower diagonal: p-value for H0: difference = 0  
##   
## MAE   
## knn svm mars   
## knn 0.12913 0.09722  
## svm 5.720e-07 -0.03190  
## mars 3.194e-05 0.7201   
##   
## RMSE   
## knn svm mars   
## knn 0.14420 0.09747  
## svm 9.119e-07 -0.04673  
## mars 2.690e-05 0.3419   
##   
## Rsquared   
## knn svm mars   
## knn -0.10445 -0.07094  
## svm 2.125e-06 0.03350  
## mars 0.0006429 0.5008691

**b.** The variable importance, 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? - Yes, ManufacturingProcessXX dominates the list. There are 4 BiologicalMaterials that rank 2,6,8,9. - ManufacturingProcess32 is the most important predictor in both pls and svm. The remaining dominant predictors are also very similar and maintain similar order of importance.

dotPlot(varImp(plsModel),main="plsModel", top=10)



dotPlot(varImp(svmModel),main="svmModel", top=10)



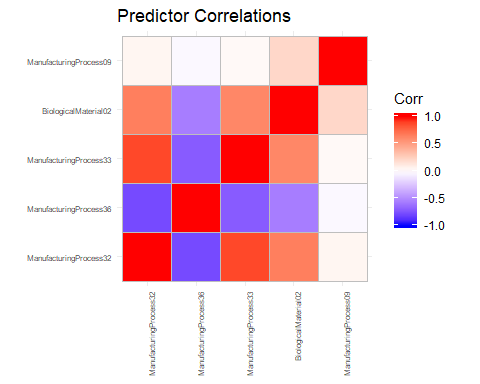
**c.** Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model.Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

* Explore the correlations between the top svm predictors (“ManufacturingProcess32”,“ManufacturingProcess36”, “ManufacturingProcess33”, “BiologicalMaterial02”,“ManufacturingProcess09”)
* “ManufacturingProcess32” and “ManufacturingProcess33” have a correlation value of .87 which means they are likely providing same information and are redundant in the model
* The four of the five top predictors show low to moderate correlation with the response variable
* Next predictor and plot how the response varies as a function of this value. Build a dataframe with variation by each of the five dataframes

#y=yield\_training  
top.pred.svm <- processPredictors\_training %>%  
 select(c("ManufacturingProcess32","ManufacturingProcess36", "ManufacturingProcess33", "BiologicalMaterial02","ManufacturingProcess09"))  
  
print("Cor() TOP Pred")

## [1] "Cor() TOP Pred"

ggcorrplot(cor(top.pred.svm))+  
 theme(axis.text.x=element\_text(size=rel(.7), angle=90, hjust=1),  
 axis.text.y = element\_text(size=rel(.7), hjust=1))+  
 ggtitle("Predictor Correlations")



reshape2::melt(cor(top.pred.svm))%>%  
 rename(Predictor1 = Var1, Predictor2 = Var2, CorrelationValue = value)%>%  
 filter(CorrelationValue != 1)%>%   
 filter (! duplicated(CorrelationValue))

## Predictor1 Predictor2 CorrelationValue  
## 1 ManufacturingProcess36 ManufacturingProcess32 -0.77629083  
## 2 ManufacturingProcess33 ManufacturingProcess32 0.87294630  
## 3 BiologicalMaterial02 ManufacturingProcess32 0.64617672  
## 4 ManufacturingProcess09 ManufacturingProcess32 0.04626312  
## 5 ManufacturingProcess33 ManufacturingProcess36 -0.71488392  
## 6 BiologicalMaterial02 ManufacturingProcess36 -0.55764138  
## 7 ManufacturingProcess09 ManufacturingProcess36 -0.02562218  
## 8 BiologicalMaterial02 ManufacturingProcess33 0.61165716  
## 9 ManufacturingProcess09 ManufacturingProcess33 0.02846168  
## 10 ManufacturingProcess09 BiologicalMaterial02 0.20623385

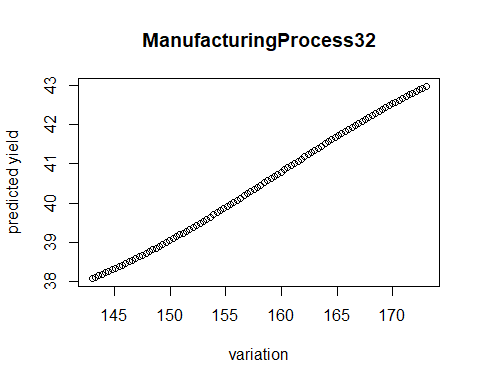
print("Cor() Against Yield")

## [1] "Cor() Against Yield"

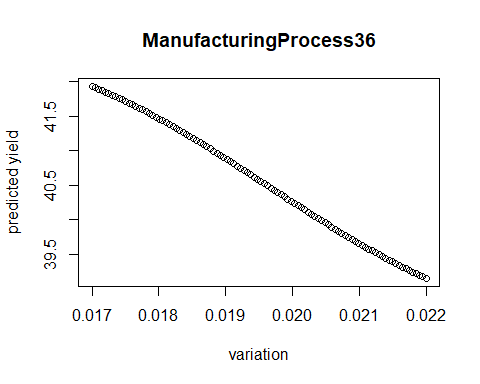
cor(top.pred.svm, yield\_training)

## [,1]  
## ManufacturingProcess32 0.6490227  
## ManufacturingProcess36 -0.5494493  
## ManufacturingProcess33 0.5100480  
## BiologicalMaterial02 0.4752546  
## ManufacturingProcess09 0.4640182

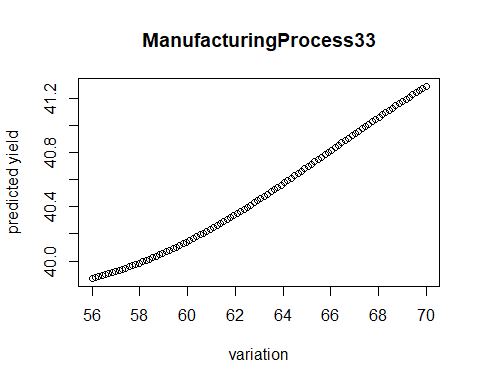
plot.funct <- function(processPredictors, predictor){   
 p\_range = range( processPredictors[,predictor] )  
  
 variation = seq( from=p\_range[1], to=p\_range[2], length.out=100 )  
 mean\_predictor\_values = apply( processPredictors, 2, mean )  
   
 # build a dataframe with variation in only one dimension (for this part we pick ManufacturingProcess32)  
   
 newdata = repmat( as.double(mean\_predictor\_values), length(variation), 1 )  
 newdata = data.frame( newdata )  
  
 colnames( newdata ) = colnames( processPredictors )  
 newdata[,predictor] = variation  
 xs = variation  
 y\_hat = predict( svmModel, newdata=as.matrix(newdata) )  
 return(plot( xs, y\_hat, xlab='variation', ylab='predicted yield' , main = predictor))  
  
   
}  
plot.funct(processPredictors, "ManufacturingProcess32")



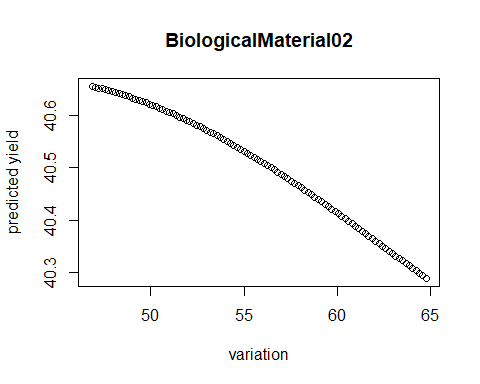
plot.funct(processPredictors, "ManufacturingProcess36")



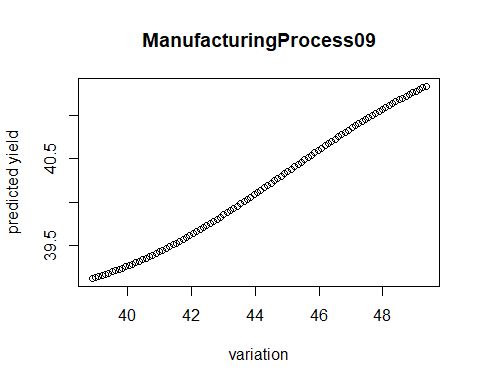
plot.funct(processPredictors, "ManufacturingProcess33")



plot.funct(processPredictors, "BiologicalMaterial02")



plot.funct(processPredictors, "ManufacturingProcess09")



featurePlot(top.pred.svm, yield\_training)

