HW6.3

Alejandro Osborne

April 23, 2019

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(AppliedPredictiveModeling)

## Warning: package 'AppliedPredictiveModeling' was built under R version  
## 3.5.3

library(elasticnet)

## Warning: package 'elasticnet' was built under R version 3.5.2

## Loading required package: lars

## Warning: package 'lars' was built under R version 3.5.2

## Loaded lars 1.2

library(pracma)

## Warning: package 'pracma' was built under R version 3.5.3

# Part (a): Data Loading

# 

set.seed(0)  
data(ChemicalManufacturingProcess)  
  
processPredictors = ChemicalManufacturingProcess[,2:58]  
yield = ChemicalManufacturingProcess[,1]  
  
n\_sample = dim(processPredictors)[1]  
n\_feature = dim(processPredictors)[2]

# Part (b): Fill in missing values where we have NAs with the median over the non-NA values:

# 

replacements = sapply( processPredictors, median, na.rm=TRUE )  
for( ci in 1:n\_feature ){  
 bad\_inds = is.na( processPredictors[,ci] )  
 processPredictors[bad\_inds,ci] = replacements[ci]  
}  
  
# Look for non-variance features:  
#   
zero\_cols = nearZeroVar( processPredictors )  
print( sprintf("Found %d zero variance columns from %d",length(zero\_cols), dim(processPredictors)[2] ) )

## [1] "Found 1 zero variance columns from 57"

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

# Part (c): Split this data into training and testing sets:

# 

training = createDataPartition( yield, p=0.8 )  
  
pPrdctrs\_training = processPredictors[training$Resample1,]  
yld\_training = yield[training$Resample1]  
  
pPrdctrs\_testing = processPredictors[-training$Resample1,]  
yld\_testing = yield[-training$Resample1]

# Build some linear models and predict the performance on the testing data set:

# 

set.seed(0)  
pls\_model = train( pPrdctrs\_training, yld\_training, method="pls",  
 # the default tuning grid evaluates component 1  
 tuneLength=40,   
 preProcess=c("center","scale"), trControl=trainControl(method="repeatedcv",repeats=5) )  
  
y\_hat = predict( pls\_model, newdata=pPrdctrs\_testing )  
r2\_pls = cor(y\_hat,yld\_testing,method="pearson")^2  
rmse\_pls = sqrt( mean( (y\_hat-yld\_testing)^2 ) )  
print( sprintf( "%-10s: Testing R^2= %10.6f; RMSE= %10.6f", "PLS", r2\_pls, rmse\_pls ) )

## [1] "PLS : Testing R^2= 0.655223; RMSE= 1.044573"

# Will try Enet, PLS and some other models:

# 

enetGrid = expand.grid(.lambda=seq(0,1,length=20), .fraction=seq(0.05, 1.0, length=20))  
set.seed(0)  
enet\_model = train( pPrdctrs\_training, yld\_training, method="enet",  
 # fit the model over many penalty values  
 tuneGrid = enetGrid,  
 preProcess=c("center","scale"), trControl=trainControl(method="repeatedcv",repeats=5) )  
y\_hat = predict( enet\_model, newdata=pPrdctrs\_testing )  
r2\_enet = cor(y\_hat,yld\_testing,method="pearson")^2  
rmse\_enet = sqrt( mean( (y\_hat-yld\_testing)^2 ) )  
print( sprintf( "%-10s: Testing R^2= %10.6f; RMSE= %10.6f", "ENET", r2\_enet, rmse\_enet ) )

## [1] "ENET : Testing R^2= 0.643260; RMSE= 1.072692"

set.seed(0)  
lm\_model = train( pPrdctrs\_training, yld\_training, method="lm", preProcess=c("center","scale"), trControl=trainControl(method="repeatedcv",repeats=5) )

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading  
  
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading

y\_hat = predict( lm\_model, newdata=pPrdctrs\_testing )

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient  
## fit may be misleading

r2\_lm = cor(y\_hat,yld\_testing,method="pearson")^2  
rmse\_lm = sqrt( mean( (y\_hat-yld\_testing)^2 ) )  
print( sprintf( "%-10s: Testing R^2= %10.6f; RMSE= %10.6f", "LM", r2\_lm, rmse\_lm ) )

## [1] "LM : Testing R^2= 0.583151; RMSE= 1.298208"

# RLM does not allow for single predictor covar-matrix so we’re going to try with PCA:

# 

set.seed(0)  
rlm\_model = train( pPrdctrs\_training, yld\_training, method="rlm", preProcess=c("pca"), trControl=trainControl(method="repeatedcv",repeats=5) )

## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps  
  
## Warning in rlm.default(x, y, weights, method = method, wt.method =  
## wt.method, : 'rlm' failed to converge in 20 steps

y\_hat = predict( rlm\_model, newdata=pPrdctrs\_testing )  
r2\_rlm = cor(y\_hat,yld\_testing,method="pearson")^2  
rmse\_rlm = sqrt( mean( (y\_hat-yld\_testing)^2 ) )  
print( sprintf( "%-10s: Testing R^2= %10.6f; RMSE= %10.6f", "RLM", r2\_rlm, rmse\_rlm ) )

## [1] "RLM : Testing R^2= 0.618537; RMSE= 1.101150"

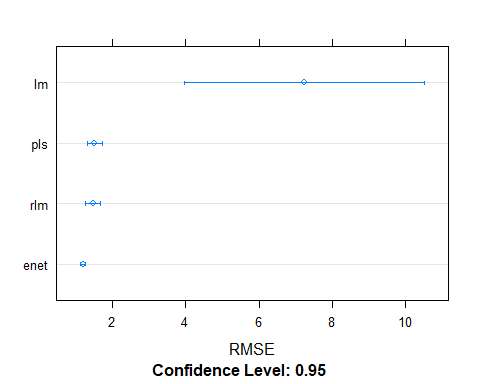
# Compare the given models using resamples

# 

resamp = resamples( list(pls=pls\_model,enet=enet\_model,lm=lm\_model,rlm=rlm\_model) )  
print( summary(resamp) )

##   
## Call:  
## summary.resamples(object = resamp)  
##   
## Models: pls, enet, lm, rlm   
## Number of resamples: 50   
##   
## MAE   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## pls 0.5725218 0.9167082 1.0633529 1.099579 1.206446 2.006919 0  
## enet 0.5592684 0.8348662 0.9524113 0.955409 1.092735 1.390613 0  
## lm 0.5527963 1.1392172 1.2978669 2.828540 3.295074 12.309195 0  
## rlm 0.5546172 0.8643729 1.0699301 1.101043 1.228147 2.182280 0  
##   
## RMSE   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## pls 0.7329661 1.096611 1.294611 1.508321 1.574285 4.681949 0  
## enet 0.7847442 1.007082 1.142443 1.195307 1.352544 2.083418 0  
## lm 0.8058616 1.328531 1.661413 7.245911 8.226307 44.507698 0  
## rlm 0.6770451 1.076250 1.322191 1.471448 1.538886 4.702059 0  
##   
## Rsquared   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## pls 0.059374498 0.39775653 0.5402962 0.5306939 0.6635787 0.8958491 0  
## enet 0.041106670 0.50714325 0.6362282 0.6124197 0.7270356 0.9126073 0  
## lm 0.001817632 0.02169739 0.4137607 0.3547662 0.5846000 0.8325257 0  
## rlm 0.022058440 0.43247784 0.5381863 0.5305493 0.6692361 0.8641249 0

dotplot( resamp, metric="RMSE" )



print( 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   
## pls enet lm rlm   
## pls 0.144170 -1.728961 -0.001464  
## enet 0.0008919 -1.873131 -0.145634  
## lm 0.0010173 0.0007891 1.727498  
## rlm 1.0000000 0.0003114 0.0013879   
##   
## RMSE   
## pls enet lm rlm   
## pls 0.31301 -5.73759 0.03687  
## enet 0.006259 -6.05060 -0.27614  
## lm 0.003519 0.003134 5.77446  
## rlm 1.000000 0.006050 0.004185   
##   
## Rsquared   
## pls enet lm rlm   
## pls -0.0817258 0.1759277 0.0001446  
## enet 0.001454 0.2576535 0.0818704  
## lm 9.714e-07 1.401e-09 -0.1757831  
## rlm 1.000000 1.297e-05 2.665e-05

# Part (e): evaluating coefficients selected by the optimal model which I believe to be Elastic Net:

# 

enet\_base\_model = enet( x=as.matrix(pPrdctrs\_training), y=yld\_training, lambda=0.5263158, normalize=TRUE )  
enet\_coefficients = predict( enet\_base\_model, newx=as.matrix(pPrdctrs\_testing), s=0.35, mode="fraction", type="coefficients" )  
  
non\_zero = enet\_coefficients$coefficients != 0  
enet\_coefficients$coefficients[ non\_zero ]

## BiologicalMaterial02 BiologicalMaterial03 BiologicalMaterial04   
## 2.680049e-02 1.027368e-02 3.735975e-04   
## BiologicalMaterial06 BiologicalMaterial08 BiologicalMaterial11   
## 3.237699e-02 6.511523e-03 5.715041e-04   
## ManufacturingProcess06 ManufacturingProcess09 ManufacturingProcess13   
## 3.607731e-02 2.009979e-01 -2.938595e-01   
## ManufacturingProcess15 ManufacturingProcess17 ManufacturingProcess32   
## 4.141848e-04 -1.548674e-01 1.101527e-01   
## ManufacturingProcess34 ManufacturingProcess36   
## 2.235127e+00 -2.948923e+02

# Part (f): The relationships between the top predictors and the response:

# 

# To do this we will pick a predictor and plot the responce as a function of this value

# 

p\_range = range( processPredictors$ManufacturingProcess32 )  
variation = seq( from=p\_range[1], to=p\_range[2], length.out=100 )  
mean\_predictor\_values = apply( processPredictors, 2, mean )  
  
# dataframe with variation in one dimension (ManufacturingProcess32 was choosen because of it's correlation)  
newdata = repmat( as.double(mean\_predictor\_values), length(variation), 1 )  
newdata = data.frame( newdata )  
colnames( newdata ) = colnames( processPredictors )  
newdata$ManufacturingProcess32 = variation  
  
xs = variation  
y\_hat = predict( enet\_base\_model, newx=as.matrix(newdata), s=0.35, mode="fraction", type="fit" )  
  
plot( xs, y\_hat$fit, xlab='variation in ManufacturingProcess32', ylab='predicted yield' )  
grid()

