# Module 3 Exercises

Filipp Krasovsky

5/31/2021

1. (30 points) Infrared (IR) spectroscopy technology is used to determine the chemical makeup of a substance. The theory of IR spectroscopy holds that unique molecular structures absorb IR frequencies differently. In practice a spectrometer fires a series of IR frequencies into a sample material, and the device measures the absorbance of the sample at each individual frequency. This series of measurements creates a spectrum profile which can then be used to determine the chemical makeup of the sample material.

A Tecator Infratec Food and Feed Analyzer instrument was used to analyze 215 samples of meat across 100 frequencies. In addition to an IR profile, analytical chemistry determined the percent content of water, fat, and protein for each sample. If we can establish a predictive relationship between IR spectrum and fat content, then food scientists could predict a sample's fat content with IR instead of using analytical chemistry. This would provide costs savings, since analytical chemistry is a more expensive, time-consuming process:

a. Start R and use these commands to load the data: library(caret), data(tecator)

```
#load in our dataset.
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.5
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
data(tecator)

#we are interested only in the relationship between IR absorption and fat.
predictors <- as.data.frame(absorp)
response   <- as.data.frame(endpoints[,2])
```

The matrix absorp contains the 100 absorbance values for the 215 samples, while matrix endpoints contains the percent of moisture, fat, and protein in columns 1–3, respectively.

b. In this example the predictors are the measurements at the individual frequencies. Because the frequencies lie in a systematic order (850–1,050 nm), the predictors have a high degree of correlation. Hence, the data lie in a smaller dimension than the total number of predictors (100). Use PCA to determine the effective dimension of these data. What is the effective dimension?

```
pca.predictors <- preProcess(predictors,method=c("center","scale","pca"))
pca.predictors
```

```
## Created from 215 samples and 100 variables
##
## Pre-processing:
##   - centered (100)
##   - ignored (0)
##   - principal component signal extraction (100)
##   - scaled (100)
##
## PCA needed 2 components to capture 95 percent of the variance
```

2 Components summarize the effective dimension space of the data.

c. Split the data into a training and a test set, pre-process the data, and build each variety of models described in this chapter. For those models with tuning parameters, what are the optimal values of the tuning parameter(s)?

```
#create data partition for training
set.seed(1)
trainingRows <- createDataPartition(predictors[,1],p = 0.80, list=FALSE)
trainPredict <- predictors[trainingRows,]
trainResponse<- response[trainingRows,]
testPredict  <- predictors[-trainingRows,]
testResponse <- response[-trainingRows,]

#pre-process data
pp.train <- preProcess(trainPredict,method=c("center","scale"))
pp.test  <- preProcess(testPredict,method=c("center","scale"))

trainPredict <- predict(pp.train,newdata=trainPredict)
testPredict  <- predict(pp.test,newdata=testPredict)
```

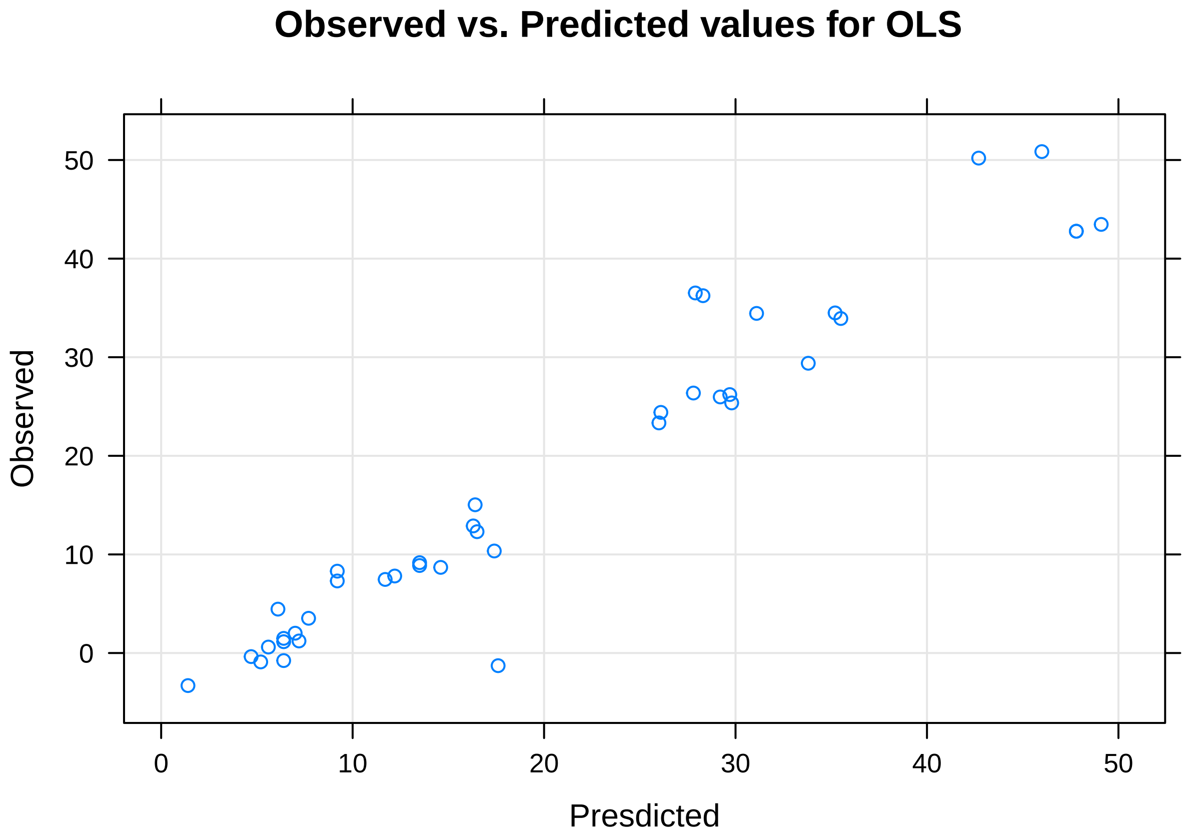
```
#model 1: OLS with 10-fold CV
ctrl <- trainControl(method="cv",number=10)
set.seed(1)
ols <- train(x = trainPredict,y=trainResponse,method="lm",trControl = ctrl)
ols.predict <- predict(ols,newdata = testPredict)
print(RMSE(ols.predict,testResponse))
```

```
## [1] 5.556543
```

```
print(caret::R2(ols.predict,testResponse))
```

```
## [1] 0.9283701
```

```
xyplot(ols.predict~testResponse,type=c("p","g"),xlab="Presdicted",ylab="Observed",main="Observed vs. Pr
```



Our linear model has an RMSE of 5.5 and an R-squared of ~93% with seemingly uncorrelated residuals and a strong correlation between observed and predicted values. Because OLS has tuning parameters, there is no optimization problem.

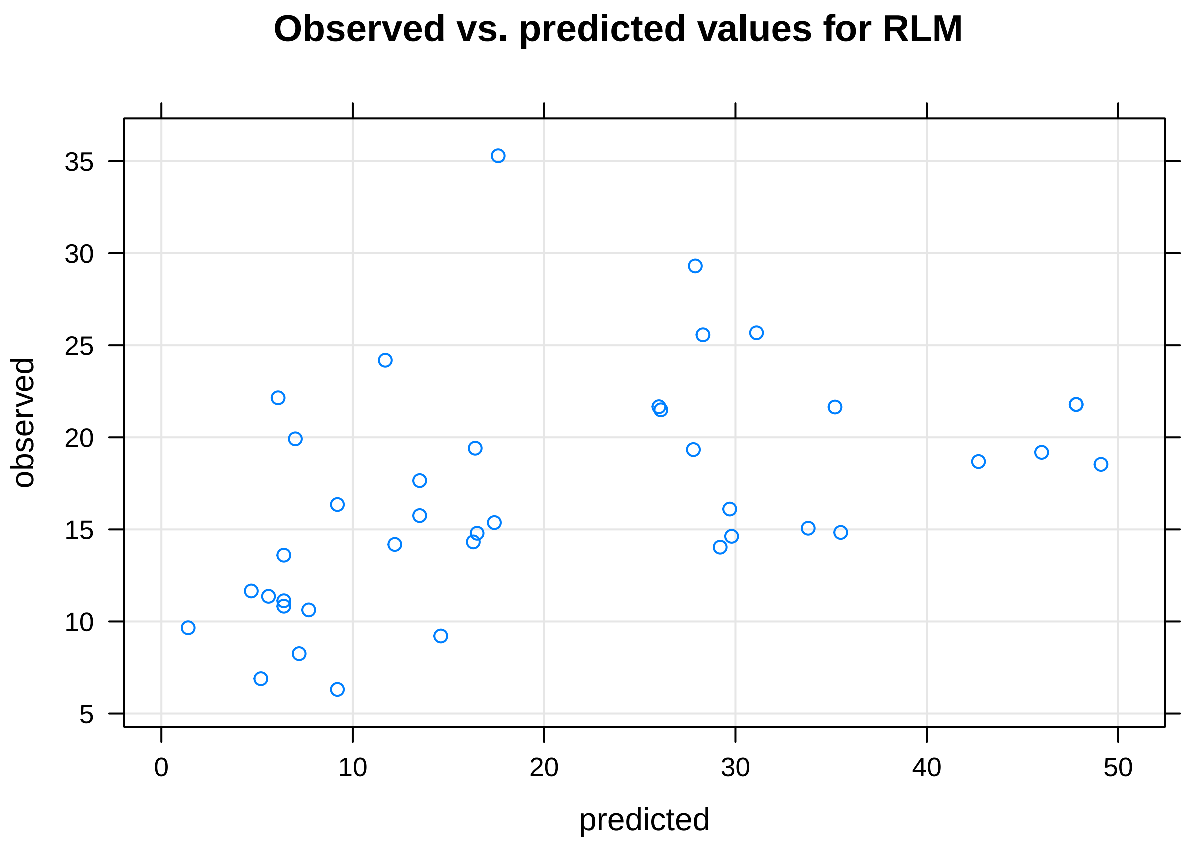
```
#model 2: rlm
set.seed(1)
rlm <- train(x = trainPredict,y=trainResponse,method="rlm", preProcess = c("center","scale","pca"), trC
rlm.predict <- predict(rlm,newdata = testPredict)
print(RMSE(rlm.predict,testResponse))
```

```
## [1] 12.86973
```

```
print(caret::R2(rlm.predict,testResponse))
```

```
## [1] 0.2063526
```

```
xyplot(rlm.predict~testResponse,type=c("p","g"),xlab="predicted",ylab="observed",main="Observed vs. pre
```



Our RLM model performed very poorly with only a 20% R-squared and an MRSE of ~13.

```
#model 3: pls
require(pls)
require(dplyr)
set.seed(1)
pls <- train(x=trainPredict,y=trainResponse, method="pls",tunelength=20,trControl = ctrl, preProc=c("ce

pls.predict <- predict(pls,newdata = testPredict) %>% as.numeric()
print(RMSE(pls.predict,testResponse))
```

```
## [1] 6.288981
```

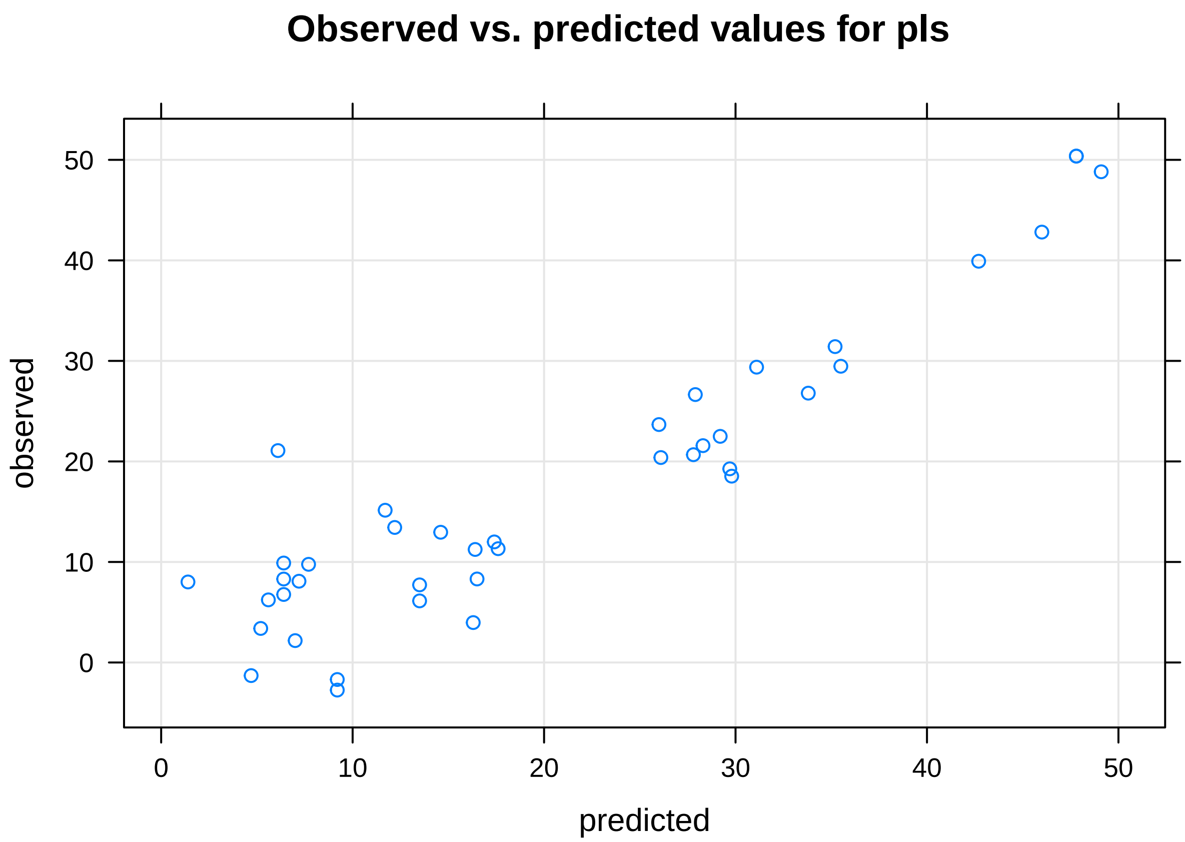
```
print(caret::R2(pls.predict,testResponse))
```

```
## [1] 0.8513959
```

```
print(paste("Optimal tuning parameter: number of components",pls$bestTune$ncomp))
```

```
## [1] "Optimal tuning parameter: number of components 3"
```

```
xyplot(pls.predict~testResponse,type=c("p","g"),xlab="predicted",ylab="observed",main="Observed vs. pre
```



Our PLS model performs with an r-squared of 85% and a MRSE of 6.28, and the optimal tuning parameter was chosen by the lowest RMSE, with ncomp = 3 havin the smallest value.

```
#penalized regression model: ridge regression
require(elasticnet)
```

```
## Loading required package: elasticnet

## Loading required package: lars

## Loaded lars 1.2
```

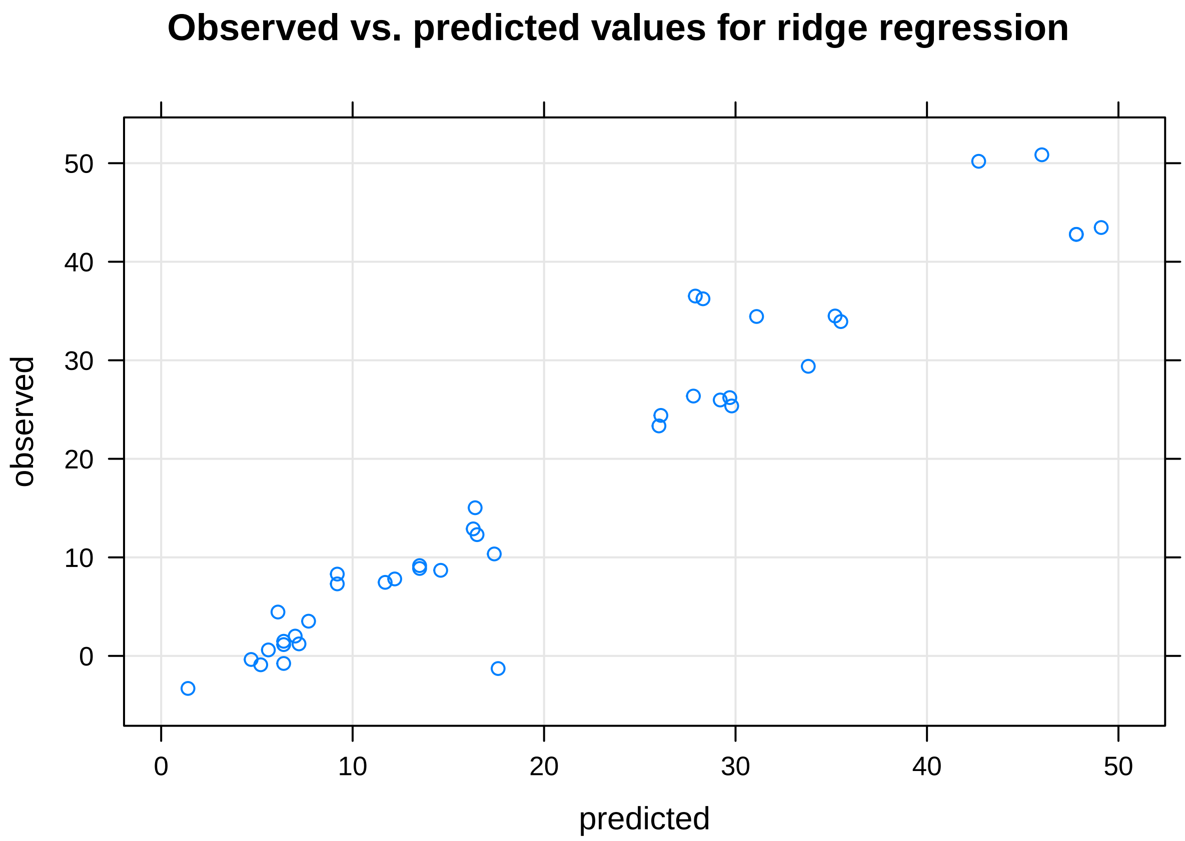
```
ridgeGrid <- data.frame(.lambda = seq(0,.1,length = 15))
set.seed(1)
ridge.reg <- train(trainPredict,trainResponse,method="ridge",tuneGrid = ridgeGrid,trControl=ctrl,prePro

ridge.predict <- predict(ridge.reg,newdata = as.matrix(testPredict))

optimal_lambda = ridge.reg$bestTune$lambda
r_squared = caret::R2(ridge.predict,testResponse)
ridge_rmse= RMSE(ridge.predict,testResponse)
print(paste("optimal tuning param (lambda):",optimal_lambda,"R2:",r_squared,"RMSE:",ridge_rmse))
```

```
## [1] "optimal tuning param (lambda): 0 R2: 0.928370449277094 RMSE: 5.55652733036382"
```

```
xyplot(ridge.predict~testResponse,type=c("p","g"),xlab="predicted",ylab="observed",main="Observed vs. pr
```

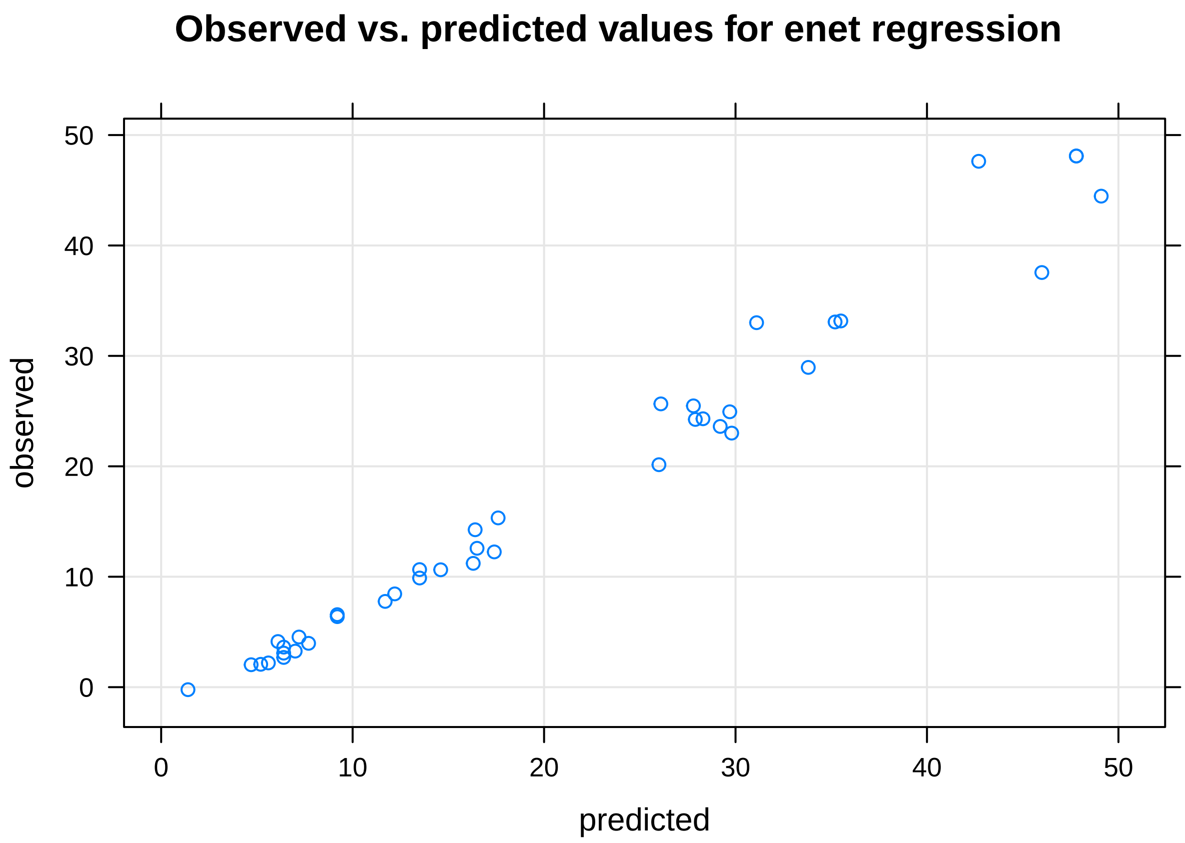


```
#penalized regression: enet regression
enetGrid <- expand.grid(.lambda=c(0,0.01,.1),.fraction=seq(0.05,1,length=20))
set.seed(1)
enet.reg <- train(trainPredict,trainResponse,method="enet",tuneGrid = enetGrid,trControl=ctrl,preProc=c
enet.predict <- predict(enet.reg,testPredict)

optimal_lambda = enet.reg$bestTune
r_squared = caret::R2(enet.predict,testResponse)
enet_rmse= RMSE(enet.predict,testResponse)
print(paste("optimal tuning param (fraction,lambda):",optimal_lambda[[1]],optimal_lambda[[2]],"R2:",r_s
```

```
## [1] "optimal tuning param (fraction,lambda): 0.05 0 R2: 0.975296090201728 RMSE: 3.82388144910209"
```

```
xyplot(enet.predict~testResponse,type=c("p","g"),xlab="predicted",ylab="observed",main="Observed vs. pre
```



d. Which model has the best predictive ability? Is any model significantly better or worse than the others?

The lasso regression seems to perform the best, while rlm seems to perform the worst.

e. Explain which model you would use for predicting the fat content of a sample.

We would use lasso regression because it has the best r-squared and the lowest RMSE.

2. (30 points) Developing a model to predict permeability (see Sect. 1.4) could save significant resources for a pharmaceutical company, while at the same time more rapidly identifying molecules that have a sufficient permeability to become a drug:

a. Start R and use these commands to load the data: library(AppliedPredictiveModeling) data(permeability) The matrix fingerprints contains the 1,107 binary molecular preditors for the 165 compounds, while permeability contains permeability response.

```
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version 4.0.5
```

```
data(permeability)
```

b. The fingerprint predictors indicate the presence or absence of substructures of a molecule and are often sparse meaning that relatively few of the molecules contain each substructure. Filter out the predictors that have low frequencies using the nearZeroVar function from the caret package. How many predictors are left for modeling?

```
require(caret)
nzv <- nearZeroVar(fingerprints)
fingerprints.df <- fingerprints[,-nzv]
length(colnames(fingerprints.df))
```

```
## [1] 388
```

We have reduced our spacefrom 1107 to 388 columns.

c. Split the data into a training and a test set, pre-process the data, and tune a PLS model. How many latent variables are optimal and what is the corresponding resampled estimate of R2?

```
#partition the training and test sets.
predictors = as.data.frame(fingerprints.df)
response   = as.data.frame(permeability)
set.seed(1)

trainingRows <- createDataPartition(predictors[,1],p = 0.80, list=FALSE)
trainPredict <- predictors[trainingRows,]
trainResponse<- response[trainingRows,]
testPredict  <- predictors[-trainingRows,]
testResponse <- response[-trainingRows,]

#pre-process data
pp.train <- preProcess(trainPredict,method=c("center","scale","BoxCox"))
pp.test  <- preProcess(testPredict,method=c("center","scale","BoxCox"))
```

```
## Warning in preProcess.default(testPredict, method = c("center", "scale", : These
## variables have zero variances: X568, X590, X591, X595
```

```
trainPredict <- predict(pp.train,newdata=trainPredict)
testPredict  <- predict(pp.test,newdata=testPredict)

paste("Training Size:",nrow(trainPredict),"Testing Size:",nrow(testPredict))
```

```
## [1] "Training Size: 132 Testing Size: 33"
```

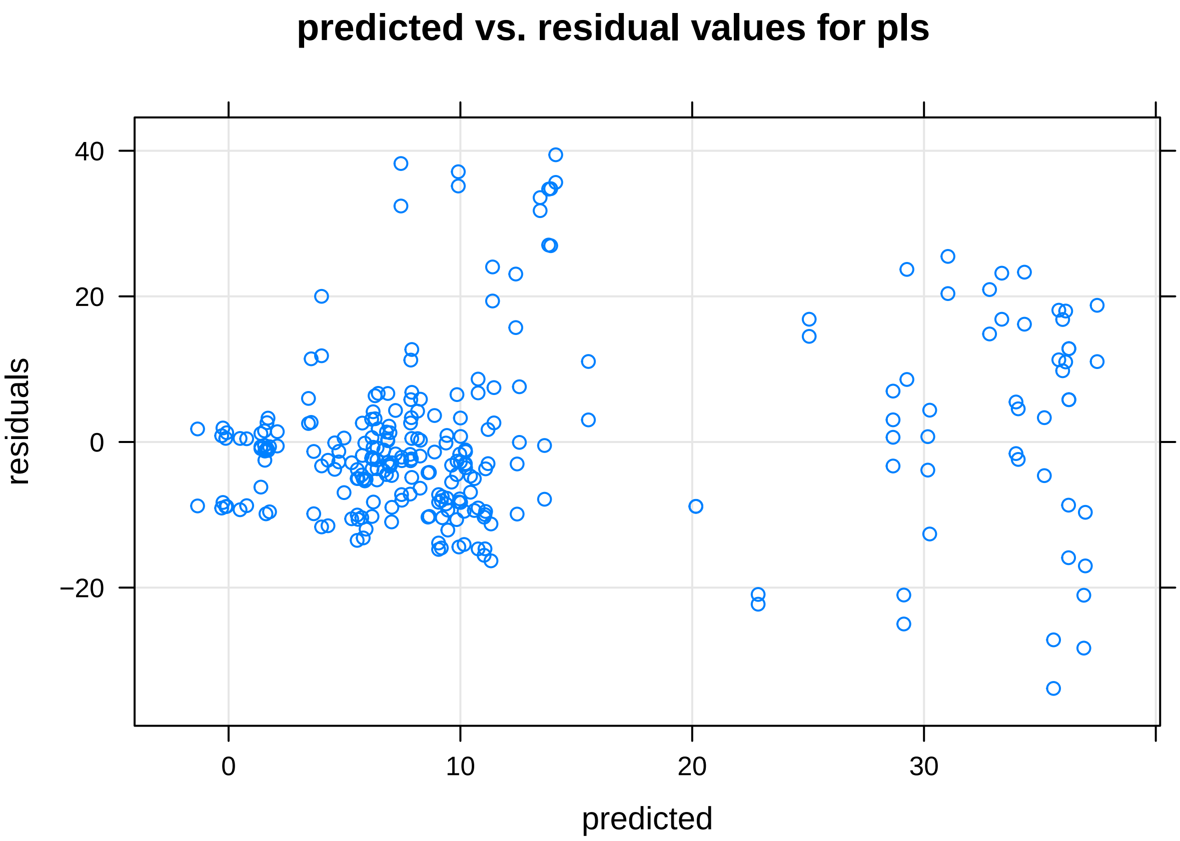
After splitting the data, we proceed with PLS:

```
#model: pls
ctrl <- trainControl(method="cv",number=10)
require(pls)
require(dplyr)
set.seed(1)
pls <- train(x=trainPredict,y=trainResponse, method="pls",tunelength=20,trControl = ctrl)

print(paste("Optimal tuning parameter: number of components",pls$bestTune$ncomp))
```

```
## [1] "Optimal tuning parameter: number of components 2"
```

```
xyplot(resid(pls)~ predict(pls),type=c("p","g"),xlab="predicted",ylab="residuals",main="predicted vs. re
```



Our optimal number of components was tuned to ncomp = 3. While technically speaking, we don't observe strong correlation between the residual and predicted values

d. Predict the response for the test set. What is the test set estimate of R2?

```
pls.predict <- predict(pls,newdata = testPredict,ncomp = 1:2) %>% as.numeric()
print(RMSE(pls.predict,testResponse))
```

```
## [1] 11.48171
```

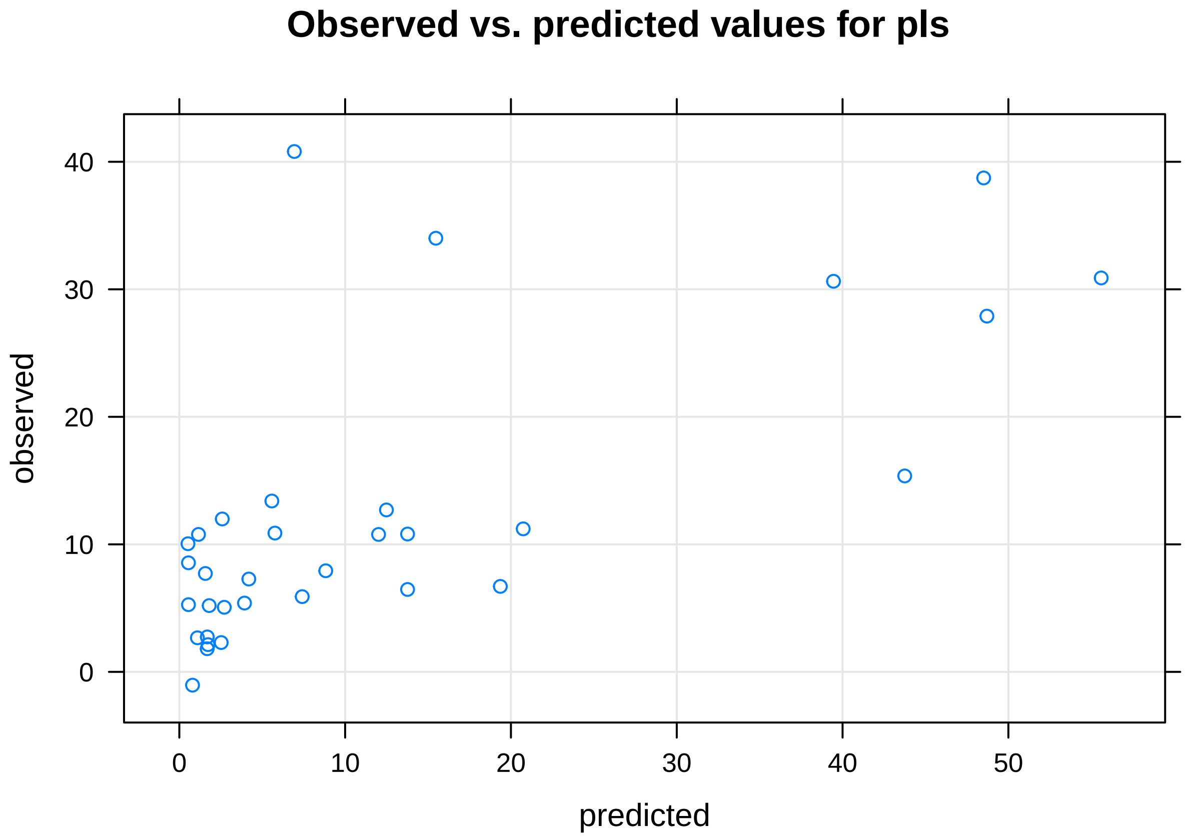
```
print(caret::R2(pls.predict,testResponse))
```

```
## [1] 0.4761436
```

```
print(paste("Optimal tuning parameter: number of components",pls$bestTune$ncomp))
```

```
## [1] "Optimal tuning parameter: number of components 2"
```

```
xyplot(pls.predict~testResponse,type=c("p","g"),xlab="predicted",ylab="observed",main="Observed vs. pred
```



The test R-squared estimate is about 47%, suggesting a very poor performance.

e. Try building other models discussed in this chapter. Do any have better predictive performance?

```
#ols: we cannot conduct ols on this model without first applying PCA.

set.seed(1)
ols <- train(x = trainPredict,y=trainResponse,method="lm",trControl = ctrl,preProcess = "pca")
ols.predict <- predict(ols,newdata = testPredict)
print(RMSE(ols.predict,testResponse))
```

```
## [1] 10.79666
```

```
print(caret::R2(ols.predict,testResponse))
```

```
## [1] 0.5444995
```

Our OLS approach doesn't suffice, but still technically outperforms pls. Next, we try lasso regression.

```
#penalized regression: Lasso regression
set.seed(1)
enetModel<- enet(x = as.matrix(trainPredict), y = trainResponse, lambda = 0, normalize = FALSE)
enetPred <- predict(enetModel, newx = as.matrix(testPredict), s = .1, mode = "fraction", type = "fit")
r_squared = caret::R2(enetPred$fit,testResponse)
ridge_rmse= RMSE(enetPred$fit,testResponse)
paste(r_squared,ridge_rmse)
```

```
## [1] "0.450433219843631 11.7617144568266"
```

We can also try the glmnet package for lasso:

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.5

## Loading required package: Matrix

## Loaded glmnet 4.1-1
```

```
require(caret)
#perform k-fold cross-validation to find optimal lambda value
cv_model <- cv.glmnet(as.matrix(trainPredict), trainResponse, alpha = 1)

#find optimal lambda value that minimizes test MSE
best_lambda <- cv_model$lambda.min
best_model <- glmnet(as.matrix(trainPredict), trainResponse, alpha = 1, lambda = best_lambda)
lasso <- predict(best_model, s = best_lambda, newx = as.matrix(testPredict))
print(caret::R2(lasso,testResponse))
```

```
##              1
## [1,] 0.4266438
```

```
print(caret::RMSE(lasso,testResponse))
```

```
## [1] 12.01184
```

We can also try an enet:

```
enetGrid <- expand.grid(.lambda = c(0, 0.01, .1), .fraction = seq(.05, 1, length = 20))
set.seed(1)
enetTune <- train(trainPredict,trainResponse,method = "enet",tuneGrid = enetGrid,trControl = ctrl)
```

```
## Warning: model fit failed for Fold03: lambda=0.00, fraction=1 Error in if (zmin < gamhat) { : missing

## Warning: model fit failed for Fold08: lambda=0.00, fraction=1 Error in if (zmin < gamhat) { : missing

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
fraction.b = enetTune$bestTune$fraction
lambda.b = enetTune$bestTune$lambda

enetModel <- enet(x = as.matrix(trainPredict), y = trainResponse, lambda = lambda.b, normalize = FALSE)
enetPred <- predict(enetModel, newx = as.matrix(testPredict), s = 0.1, mode = "fraction", type = "fit")
caret::R2(enetPred$fit,testResponse) %>% print
```

```
## [1] 0.4549504
```

```
caret::RMSE(enetPred$fit,testResponse) %>% print
```

```
## [1] 11.82005
```

f. Would you recommend any of your models to replace the permeability laboratory experiment?

All of these models, as it stands, lack the explanatory power to reliably predict permeability. OLS could predict variation the best, but still falls well below adequate performance by R-squared measure.

3. (30 points) A chemical manufacturing process for a pharmaceutical product was discussed in Sect.1.4 of the textbook. In this problem, the objective is to understand the relationship between biological measurements of the raw materials (predictors), measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1% will boost revenue by approximately one hundred thousand dollars per batch:

a. Start R and use these commands to load the data: library(AppliedPredictiveModeling) data(chemicalManufacturingProc

```
library(AppliedPredictiveModeling)
data(ChemicalManufacturingProcess)
```

The matrix processPredictors contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. yield contains the percent yield for each run.

b. A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values.

```
#knn imputation
require(Hmisc)
```

```
## Loading required package: Hmisc
```

```
## Warning: package 'Hmisc' was built under R version 4.0.5
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

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

```
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
predictors = (ChemicalManufacturingProcess) %>% subset(select=-c(Yield)) %>% data.frame
predictors = Hmisc::impute(predictors)
response  <-ChemicalManufacturingProcess$Yield %>% as.data.frame

#remove nzv
nzv = nearZeroVar(predictors)
predictors = predictors[-nzv]
```

c. Split the data into a training and a test set, pre-process the data, and tune a model of your choice from chapter 6. What is the optimal value of the performance metric?
d. Predict the response for the test set. What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

```
#create data partition for training
set.seed(1)
trainingRows <- createDataPartition(predictors[,1],p = 0.80, list=FALSE)
trainPredict <- predictors[trainingRows,]
trainResponse<- response[trainingRows,]
testPredict  <- predictors[-trainingRows,]
testResponse <- response[-trainingRows,]

#pre-process data
pp.train <- preProcess(trainPredict,method=c("center","scale","BoxCox"))
pp.test  <- preProcess(testPredict,method=c("center","scale","BoxCox"))

trainPredict <- predict(pp.train,newdata=trainPredict)
testPredict  <- predict(pp.test,newdata=testPredict)
```

We will tune a lasso model:

```
library(glmnet)
require(caret)
#perform k-fold cross-validation to find optimal lambda value
cv_model <- cv.glmnet(as.matrix(trainPredict), trainResponse, alpha = 0.5)

#find optimal lambda value that minimizes test MSE
best_lambda <- cv_model$lambda.min
best_model <- glmnet(as.matrix(trainPredict), trainResponse, alpha = 0.5, lambda = best_lambda)
lasso <- predict(best_model, s = best_lambda, newx = as.matrix(testPredict))

#print test metrics
print(caret::R2(lasso,testResponse))
```

```
##              1
## [1,] 0.5042932
```

```
print(caret::RMSE(lasso,testResponse))
```

```
## [1] 1.285941
```

```
#print train metrics
print(cv_model$cvm %>% min %>% sqrt)
```

```
## [1] 1.114403
```

Our predicted RMSE is 1.13 and our test RMSE is 1.277. Out R-squared is very low.

e. Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

```
best_model$beta
```

```
## 56 x 1 sparse Matrix of class "dgCMatrix"
##                                   s0
## BiologicalMaterial01    .
## BiologicalMaterial02    .
## BiologicalMaterial03    .
## BiologicalMaterial04    .
## BiologicalMaterial05    0.102555005
## BiologicalMaterial06    0.036113013
## BiologicalMaterial08    .
## BiologicalMaterial09    .
## BiologicalMaterial10   -0.002440517
## BiologicalMaterial11    .
## BiologicalMaterial12    0.014027266
## ManufacturingProcess01  .
## ManufacturingProcess02  .
## ManufacturingProcess03  .
## ManufacturingProcess04  .
## ManufacturingProcess05  .
## ManufacturingProcess06  0.018944497
## ManufacturingProcess07  .
## ManufacturingProcess08  .
## ManufacturingProcess09  0.475562435
## ManufacturingProcess10  .
## ManufacturingProcess11  .
## ManufacturingProcess12  .
## ManufacturingProcess13 -0.276093105
## ManufacturingProcess14  .
## ManufacturingProcess15  0.035798486
## ManufacturingProcess16  .
## ManufacturingProcess17 -0.328087691
## ManufacturingProcess18  .
## ManufacturingProcess19  .
## ManufacturingProcess20  .
## ManufacturingProcess21  .
## ManufacturingProcess22  .
## ManufacturingProcess23  .
## ManufacturingProcess24  .
## ManufacturingProcess25  .
## ManufacturingProcess26  .
## ManufacturingProcess27  .
## ManufacturingProcess28 -0.068713312
## ManufacturingProcess29  .
## ManufacturingProcess30  .
## ManufacturingProcess31  .
## ManufacturingProcess32  0.876595090
## ManufacturingProcess33  .
## ManufacturingProcess34  .
## ManufacturingProcess35  .
## ManufacturingProcess36 -0.256741434
## ManufacturingProcess37 -0.121031701
## ManufacturingProcess38  .
## ManufacturingProcess39  0.062611099
## ManufacturingProcess40  .
## ManufacturingProcess41  .
## ManufacturingProcess42  .
## ManufacturingProcess43  0.015310322
## ManufacturingProcess44  .
## ManufacturingProcess45  .
```

As we can see, 3 of the predictors are biological and the rest are manufcating based.

f. Explore the relationships between each of the top predictors and the response. How could this information be helpful in improving yield in future runs of the manufacturing process?

```
predictors.top <- subset(ChemicalManufacturingProcess,select=c(BiologicalMaterial05,BiologicalMaterial0
predictors.top = impute(predictors.top)

#a. visualize.
require(corrplot)
```

```
## Loading required package: corrplot
```

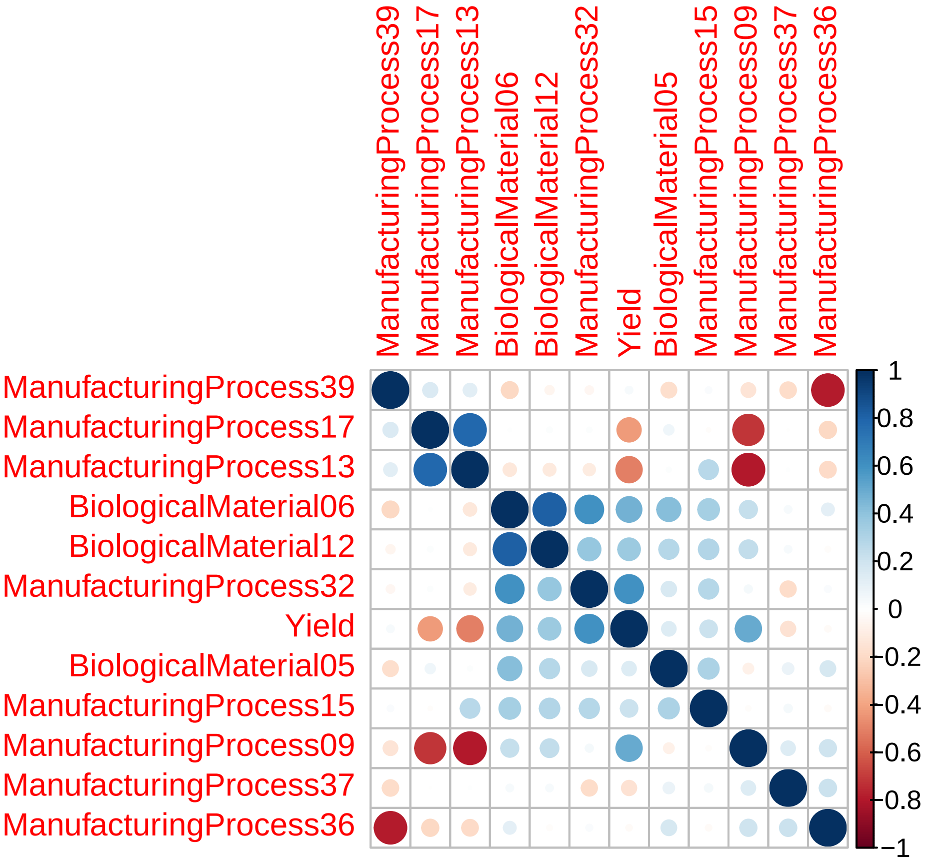
```
## Warning: package 'corrplot' was built under R version 4.0.5
```

```
## corrplot 0.88 loaded
```

```
##
## Attaching package: 'corrplot'
```

```
## The following object is masked from 'package:pls':
##
##     corrplot
```

```
#extract predictors into correlogram
predictors.cor = cor(predictors.top)
corrplot(predictors.cor,order="hclust",)
```



Yield seems to have a moderate correlation with ManufcaturingProcess33 and 37, while having a weak negative correlation with Manufacturing processes 17 and 13. It's also evident that some of the manufacturing processes are very strongly correlated with each other. Overall, the trend that informs the modeling process is that manufacturing processes are negatively associated with yield, while biological processes tend to be positively associated with yield.