Exercise 3

Austin Vanderlyn ajl745

7/2/2022

Exercise 3

3.a

```
library(caret)
```

Start R and use these commands to load the data;

```
## Warning: package 'caret' was built under R version 4.1.2

## Loading required package: ggplot2

## Loading required package: lattice
```

```
data(tecator)
```

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

3.b

Use PCA to determine the effective dimension of these data. What is the effective dimension? Run PCA on absorb;

```
PCA.ab = prcomp(absorp,
                center = TRUE,
                scale = TRUE)
```

To determine the principal components, calculate variance explained by each component;

```
var = PCA.ab$sdev^2/sum(PCA.ab$sdev^2)*100
head(var)
```

```
## [1] 98.626192582  0.969705229  0.279324276  0.114429868  0.006460911
## [6]  0.002624591
```

This shows that almost all of the variance is explained by the first component, which is much less than the number of predictors.

3.c

Split the data into a training and a test set for the response of the percentage of moisture, 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 parameters? Split data;

```
set.seed(123)
ab = data.frame(absorp)
split = createDataPartition(endpoints[, 3],
                            p = 0.8,
                            list = FALSE)

absorpTrain = ab[split,]
absorpTest = ab[-split,]
endTrain = endpoints[split, 3]
endTest = endpoints[-split,3]
```

Train control;

```
ctrl = trainControl(method = "repeatedcv",
                    repeats = 10)
```

Linear Model;

```
set.seed(123)
ab.lm = train(absorpTrain, endTrain,
              method = "lm",
              trControl = ctrl)
ab.lm
```

```
## Linear Regression
##
## 174 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 155, 156, 157, 158, 158, 156, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.140373  0.8744634  0.7527115
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

PLS model;

```
set.seed(123)
pls.lm = train(absorpTrain, endTrain,
              method = "pls",
```

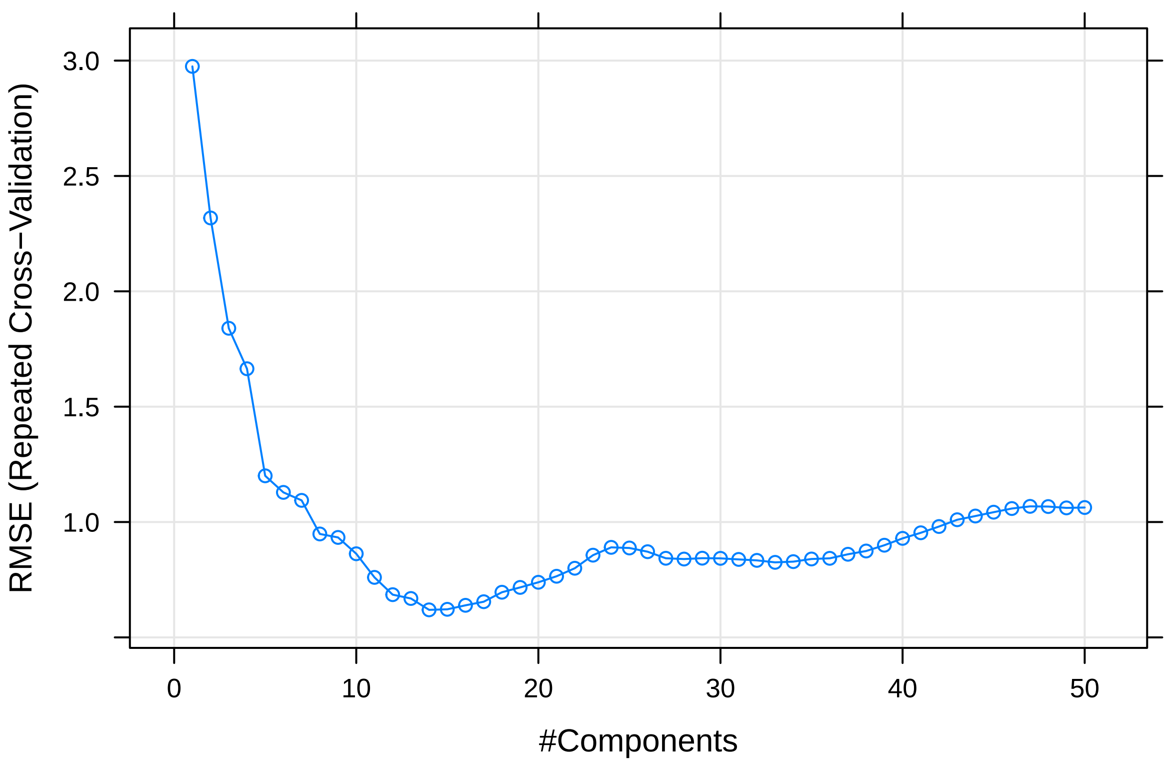
```r
              tuneGrid = expand.grid(ncomp = 1:50),
              trControl = ctrl)
pls.lm
```

```
## Partial Least Squares
##
## 174 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 155, 156, 157, 158, 158, 156, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared    MAE
##    1     2.9750800  0.09488722  2.5615013
##    2     2.3179579  0.44685688  1.8623012
##    3     1.8398275  0.63345416  1.3692728
##    4     1.6644213  0.70502969  1.2781952
##    5     1.2003331  0.86203579  0.9621822
##    6     1.1285207  0.87804564  0.9095925
##    7     1.0938298  0.88276436  0.8704333
##    8     0.9482667  0.91088869  0.7674513
##    9     0.9332279  0.91363636  0.7521155
##   10     0.8624612  0.92611872  0.6972237
##   11     0.7600335  0.94142809  0.6069046
##   12     0.6851116  0.95441017  0.5288769
##   13     0.6686291  0.95693091  0.5194797
##   14     0.6193285  0.96255233  0.4838405
##   15     0.6216509  0.96208991  0.4847157
##   16     0.6389651  0.95979315  0.4927852
##   17     0.6546149  0.95802379  0.5019089
##   18     0.6956852  0.95216222  0.5190362
##   19     0.7166207  0.94868792  0.5259514
##   20     0.7387892  0.94578306  0.5323318
##   21     0.7649537  0.94143605  0.5448489
##   22     0.7996920  0.93349564  0.5555620
##   23     0.8563171  0.91989276  0.5794565
##   24     0.8902504  0.91226350  0.5888527
##   25     0.8873935  0.91527363  0.5869942
##   26     0.8713363  0.92122810  0.5749737
##   27     0.8428920  0.92882389  0.5594604
##   28     0.8395109  0.93032874  0.5563496
##   29     0.8432169  0.93025853  0.5593709
##   30     0.8425676  0.92988851  0.5594057
##   31     0.8378858  0.92917162  0.5617852
##   32     0.8337605  0.92819550  0.5634263
##   33     0.8251919  0.92920361  0.5607199
##   34     0.8282657  0.92825341  0.5648284
##   35     0.8397615  0.92707244  0.5749504
##   36     0.8426413  0.92731923  0.5815054
##   37     0.8602956  0.92461146  0.5955255
##   38     0.8744062  0.92260061  0.6047010
```

```
## 39 0.8994198 0.91855943 0.6200059
## 40 0.9291232 0.91488926 0.6364572
## 41 0.9533327 0.91032384 0.6454746
## 42 0.9805466 0.90546674 0.6609130
## 43 1.0099019 0.89972350 0.6749573
## 44 1.0260445 0.89710766 0.6837467
## 45 1.0426795 0.89377608 0.6911078
## 46 1.0584886 0.89061396 0.7006850
## 47 1.0678255 0.88930843 0.7065204
## 48 1.0670568 0.88875560 0.7060402
## 49 1.0612248 0.88951519 0.7034587
## 50 1.0629138 0.88921164 0.7033770
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 14.
```

Plot PLS model;

```
plot(pls.lm)
```



PCR model;

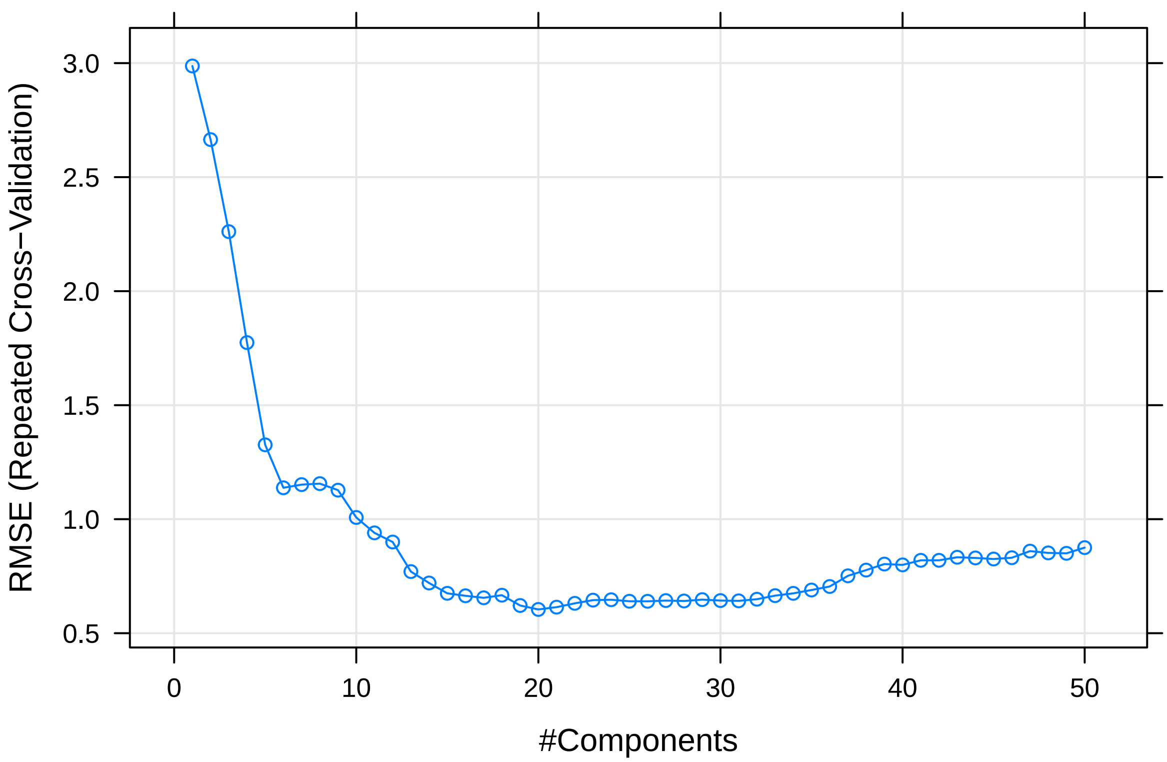
```
set.seed(123)
pcr.lm = train(absorpTrain, endTrain,
               method = "pcr",
               preProcess = c("center", "scale"),
```

```r
               tuneGrid = expand.grid(ncomp = 1:50),
               trControl = ctrl)
pcr.lm
```

```
## Principal Component Analysis
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 155, 156, 157, 158, 158, 156, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared    MAE
##    1     2.9876383  0.08692979  2.5764043
##    2     2.6647438  0.25945783  2.1813756
##    3     2.2611373  0.46032435  1.8001365
##    4     1.7745210  0.66581283  1.3486130
##    5     1.3260889  0.82727034  1.0542801
##    6     1.1376274  0.87646903  0.9208922
##    7     1.1515389  0.87295996  0.9262657
##    8     1.1559237  0.87155497  0.9314669
##    9     1.1272705  0.87890711  0.9145246
##   10     1.0075150  0.90127323  0.8111784
##   11     0.9400181  0.91077941  0.7608175
##   12     0.8998420  0.91867403  0.7278282
##   13     0.7702267  0.94222657  0.5923559
##   14     0.7200592  0.94985803  0.5595222
##   15     0.6748664  0.95634264  0.5250725
##   16     0.6640860  0.95757048  0.5170245
##   17     0.6551342  0.95853226  0.5135408
##   18     0.6665983  0.95757433  0.5168396
##   19     0.6214880  0.96235771  0.4824489
##   20     0.6041453  0.96426905  0.4728739
##   21     0.6141388  0.96281530  0.4813065
##   22     0.6309775  0.96068305  0.4884220
##   23     0.6449296  0.95874519  0.4961549
##   24     0.6467289  0.95865807  0.4970455
##   25     0.6400264  0.95923459  0.4943042
##   26     0.6398079  0.95931818  0.4911894
##   27     0.6429916  0.95917843  0.4950795
##   28     0.6411988  0.95942950  0.4936579
##   29     0.6471827  0.95870236  0.4976554
##   30     0.6429433  0.95946359  0.4947062
##   31     0.6418369  0.95954609  0.4940014
##   32     0.6489919  0.95863983  0.4968854
##   33     0.6645933  0.95648502  0.5042199
##   34     0.6746638  0.95504396  0.5092633
##   35     0.6892847  0.95298043  0.5163764
##   36     0.7047909  0.95064765  0.5239878
##   37     0.7516148  0.94142650  0.5444417
##   38     0.7767389  0.93629124  0.5548613
```

```
##   39      0.8031905  0.93151477  0.5630217
##   40      0.7996135  0.93167519  0.5567758
##   41      0.8195921  0.92661610  0.5617097
##   42      0.8195426  0.92565845  0.5595271
##   43      0.8330087  0.92364533  0.5657032
##   44      0.8300152  0.92482984  0.5636242
##   45      0.8252755  0.92686918  0.5635996
##   46      0.8309633  0.92684902  0.5680599
##   47      0.8599381  0.92401814  0.5834284
##   48      0.8521379  0.92595519  0.5817922
##   49      0.8502275  0.92752590  0.5860516
##   50      0.8752371  0.92291884  0.6013271
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 20.
```

```
plot(pcr.lm)
```



Ridge Regression model;

```
set.seed(123)
ptm = proc.time()
library(elasticnet)
```

```
## Warning: package 'elasticnet' was built under R version 4.1.3
```

```
## Loading required package: lars

## Warning: package 'lars' was built under R version 4.1.3

## Loaded lars 1.3
```

```
ridgeGrid = expand.grid(lambda = seq(0, .1,
                                     length = 10))

ridge.lm = train(absorpTrain, endTrain,
                 method = "ridge",
                 tuneGrid = ridgeGrid,
                 trControl = ctrl,
                 preProcess = c("center", "scale"))

ridge.lm
```

```
## Ridge Regression
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 155, 156, 157, 158, 158, 156, ...
## Resampling results across tuning parameters:
##
##   lambda      RMSE      Rsquared   MAE
##   0.00000000  1.140358  0.8744647  0.7527058
##   0.01111111  1.445364  0.7879636  1.1328244
##   0.02222222  1.575352  0.7455248  1.2162222
##   0.03333333  1.646085  0.7226408  1.2672710
##   0.04444444  1.694587  0.7072924  1.3033271
##   0.05555556  1.732194  0.6955728  1.3339823
##   0.06666667  1.763446  0.6858828  1.3630179
##   0.07777778  1.790508  0.6774585  1.3890917
##   0.08888889  1.814554  0.6698928  1.4122015
##   0.10000000  1.836287  0.6629498  1.4329859
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 0.
```

ENET Model;

```
set.seed(123)

enetGrid = expand.grid(lambda = c(0, 0.01, .1),
                       fraction = seq(.05, 1,
                                      length = 20))

enet.lm = train(absorpTrain, endTrain,
                method = "enet",
```

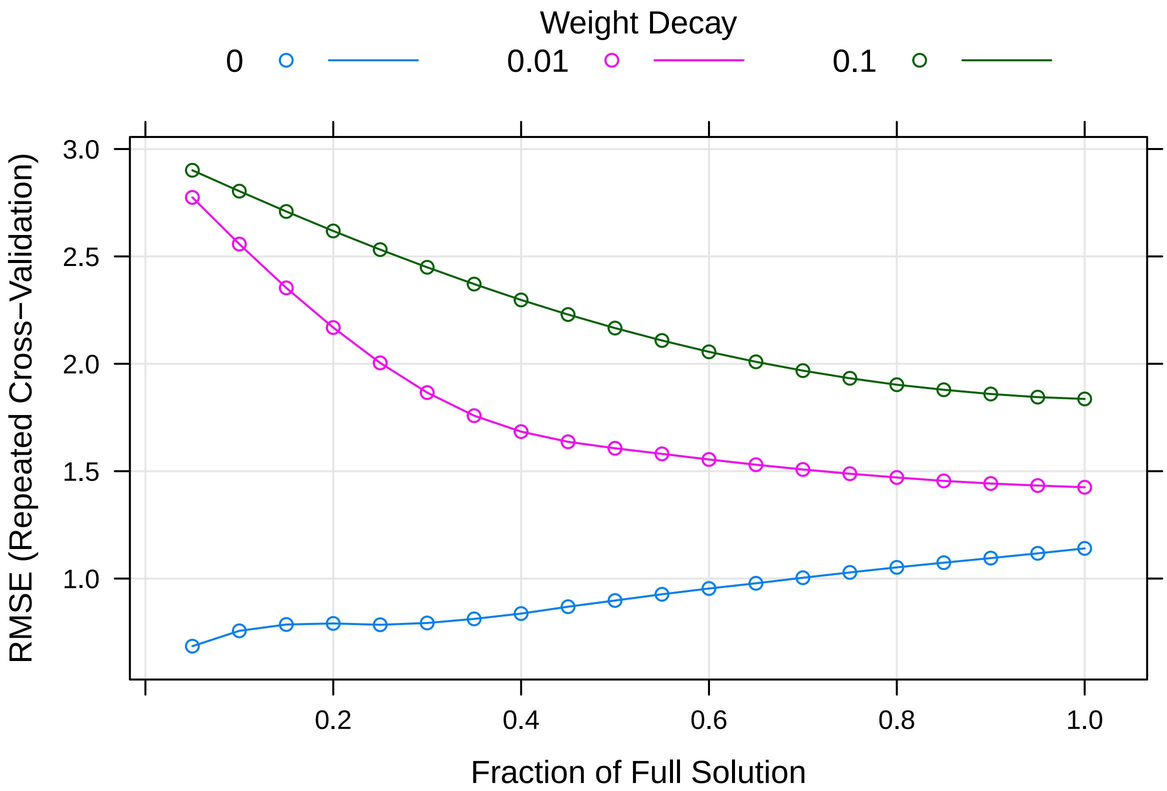
```r
                tuneGrid = enetGrid,
                trControl = ctrl,
                preProcess = c("center", "scale"))

enet.lm
```

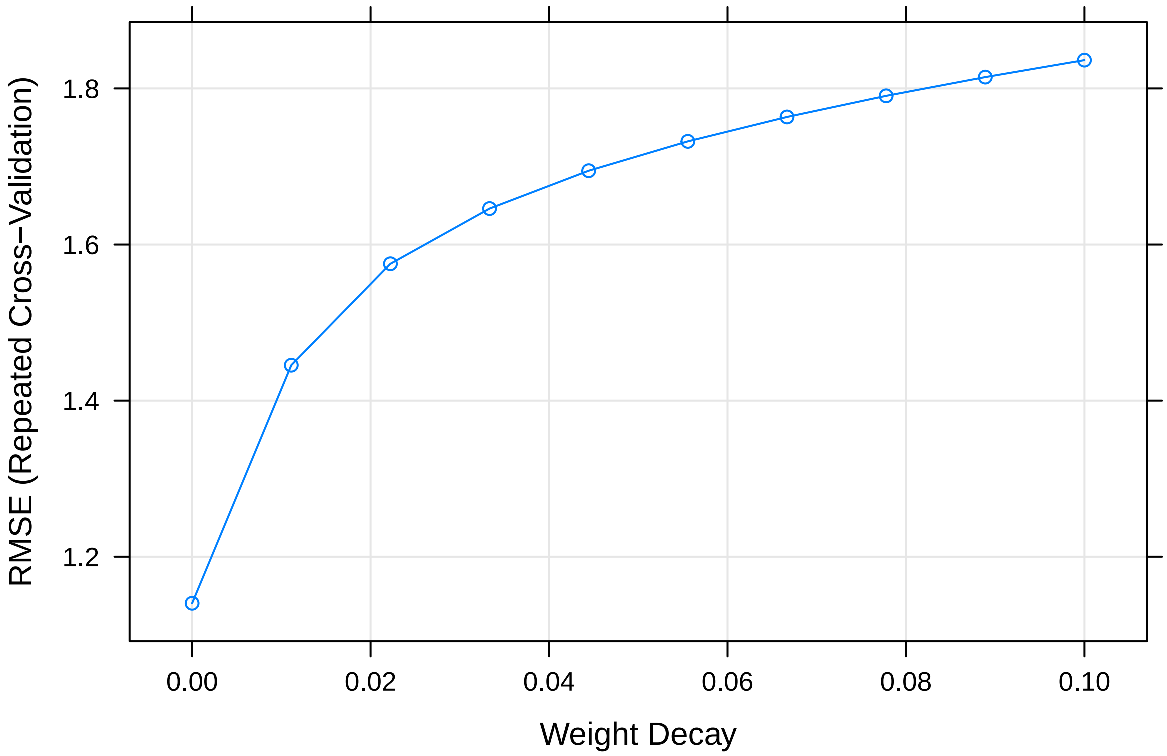
```
## Elasticnet
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 155, 156, 157, 158, 158, 156, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE       Rsquared   MAE
##   0.00    0.05      0.6850805  0.9504180  0.4924303
##   0.00    0.10      0.7566840  0.9406428  0.5252643
##   0.00    0.15      0.7860109  0.9367214  0.5339125
##   0.00    0.20      0.7910279  0.9360205  0.5320571
##   0.00    0.25      0.7848146  0.9369017  0.5310964
##   0.00    0.30      0.7931308  0.9353796  0.5380869
##   0.00    0.35      0.8123171  0.9324028  0.5507717
##   0.00    0.40      0.8369347  0.9286835  0.5662504
##   0.00    0.45      0.8690438  0.9236336  0.5842700
##   0.00    0.50      0.8978428  0.9189333  0.6017126
##   0.00    0.55      0.9268698  0.9140273  0.6194698
##   0.00    0.60      0.9536865  0.9093434  0.6358209
##   0.00    0.65      0.9780261  0.9048200  0.6499098
##   0.00    0.70      1.0039016  0.8999476  0.6648305
##   0.00    0.75      1.0285718  0.8953167  0.6796230
##   0.00    0.80      1.0521396  0.8909180  0.6942429
##   0.00    0.85      1.0739686  0.8867891  0.7085650
##   0.00    0.90      1.0952517  0.8827992  0.7227443
##   0.00    0.95      1.1174319  0.8786788  0.7375611
##   0.00    1.00      1.1403581  0.8744647  0.7527058
##   0.01    0.05      2.7747984  0.3027444  2.3945361
##   0.01    0.10      2.5575200  0.4981138  2.2013249
##   0.01    0.15      2.3535130  0.6057930  2.0139542
##   0.01    0.20      2.1687743  0.6588674  1.8377734
##   0.01    0.25      2.0041269  0.6866238  1.6702901
##   0.01    0.30      1.8658593  0.7023920  1.5170052
##   0.01    0.35      1.7584325  0.7127120  1.3962768
##   0.01    0.40      1.6840959  0.7208108  1.3125255
##   0.01    0.45      1.6368477  0.7288072  1.2663788
##   0.01    0.50      1.6065029  0.7359177  1.2406004
##   0.01    0.55      1.5805264  0.7438951  1.2214779
##   0.01    0.60      1.5544596  0.7524273  1.2037755
##   0.01    0.65      1.5299909  0.7604564  1.1874713
##   0.01    0.70      1.5079857  0.7676551  1.1729748
##   0.01    0.75      1.4882305  0.7741421  1.1600653
##   0.01    0.80      1.4705205  0.7799108  1.1485186
##   0.01    0.85      1.4548276  0.7849904  1.1384556
```

```
##   0.01    0.90      1.4425044  0.7889366  1.1307001
##   0.01    0.95      1.4331027  0.7919447  1.1249828
##   0.01    1.00      1.4251657  0.7944639  1.1201571
##   0.10    0.05      2.9010450  0.1622698  2.4992539
##   0.10    0.10      2.8038092  0.2566115  2.4114049
##   0.10    0.15      2.7092276  0.3451600  2.3254438
##   0.10    0.20      2.6184607  0.4190222  2.2439411
##   0.10    0.25      2.5316703  0.4769094  2.1645571
##   0.10    0.30      2.4493701  0.5210042  2.0891297
##   0.10    0.35      2.3711010  0.5544081  2.0162379
##   0.10    0.40      2.2972875  0.5794409  1.9455167
##   0.10    0.45      2.2293219  0.5982001  1.8781973
##   0.10    0.50      2.1662548  0.6127295  1.8130039
##   0.10    0.55      2.1085374  0.6240723  1.7506026
##   0.10    0.60      2.0556857  0.6331177  1.6917352
##   0.10    0.65      2.0091905  0.6403473  1.6395287
##   0.10    0.70      1.9682417  0.6464248  1.5933911
##   0.10    0.75      1.9325841  0.6515569  1.5523550
##   0.10    0.80      1.9024859  0.6555358  1.5165347
##   0.10    0.85      1.8791181  0.6585100  1.4882396
##   0.10    0.90      1.8592827  0.6610954  1.4637488
##   0.10    0.95      1.8447293  0.6624600  1.4444950
##   0.10    1.00      1.8362866  0.6629498  1.4329859
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were fraction = 0.05 and lambda = 0.
```

```
plot(enet.lm)
```




```
plot(ridge.lm)
```



3.d

Which model has the best predictive ability? Is any model significantly better or worse than the others? Store predictions;

```
testResults = data.frame(obs = absorpTest,
                         Linear_Regression = predict(ab.lm, absorpTest))

testResults$LRM = predict(pcr.lm, absorpTest)

testResults$PLS = predict(pls.lm, absorpTest)

testResults$PCR = predict(pcr.lm, absorpTest)

testResults$ENET = predict(enet.lm, absorpTest)

testResults$Ridge = predict(ridge.lm, absorpTest)
```

Create data frame of scores;

```
R2 = RMSE = MAE = numeric(0)

R2[1] = cor(testResults$LRM, endTest)^2
RMSE[1] = sqrt(mean((testResults$LRM - endTest)^2))
```

```r
MAE[1] = mean(abs(testResults$LRM - endTest))

R2[2] = cor(testResults$PCR, endTest)^2
RMSE[2] = sqrt(mean((testResults$PCR - endTest)^2))
MAE[2] = mean(abs(testResults$PCR - endTest))


R2[3] = cor(testResults$PLS, endTest)^2
RMSE[3] = sqrt(mean((testResults$PLS - endTest)^2))
MAE[3] = mean(abs(testResults$PLS - endTest))

R2[4] = cor(testResults$Ridge, endTest)^2
RMSE[4] = sqrt(mean((testResults$Ridge - endTest)^2))
MAE[4] = mean(abs(testResults$Ridge - endTest))

R2[5] = cor(testResults$ENET, endTest)^2
RMSE[5] = sqrt(mean((testResults$ENET - endTest)^2))
MAE[5] = mean(abs(testResults$ENET - endTest))

results = cbind(R2, RMSE, MAE)
row.names(results) = c("LRM", "PCR", "PLS", "Ridge", "ENET")
results
```

```
##              R2      RMSE       MAE
## LRM   0.9397420 0.7734525 0.5703391
## PCR   0.9397420 0.7734525 0.5703391
## PLS   0.9326235 0.8142501 0.6173506
## Ridge 0.8743995 1.1510514 0.7898330
## ENET  0.9486405 0.6589607 0.4823768
```

The models all have relatively similar R2, with the exception of the ridge regression, which has a much smaller R2, 74%. The ridge regression also has the highest RMSE and a pretty high MAE. The only other big outlier is the PLS model, which has a similar R2 but a crazy high MAE. Overall, I'd say none really stand out as being significantly better than the others, but a couple stand out as worse than the others; namely, the ridge regression and PLS.

3.e

Explain which model you would use for predicting the percentage of moisture of a sample. If I had to pick one, I would select the ENET model. It has the highest R2, the lowest RMSE and the lowest MAE. Overall, it seems very stable and accurate for predictions.