hw2

chris

2023-07-12

6.1 a)

```
library(caret)
library(earth)
data(tecator)
str(absorp)
```

```
##  num [1:215, 1:100] 2.62 2.83 2.58 2.82 2.79 ...
```

```
str(endpoints)
```

```
##  num [1:215, 1:3] 60.5 46 71 72.8 58.3 44 44 69.3 61.4 61.4 ...
```

6.1 b) In this example the predictors are the measurements at the individual frequencies. Because the frequencies lie in a systematic order (850–1,050nm), the predictors have a high degree of correlation. Hence, the data lie in a smaller dimension than the total number of predictors (215). Use PCA to determine the effective dimension of these data. What is the effective dimension?

```
pc <- prcomp(absorp, center=T,scale=T)

summary(pc)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.9311 0.9847 0.52851 0.33827 0.08038 0.05123 0.02681
## Proportion of Variance 0.9863 0.0097 0.00279 0.00114 0.00006 0.00003 0.00001
## Cumulative Proportion  0.9863 0.9960 0.99875 0.99990 0.99996 0.99999 0.99999
##                            PC8      PC9     PC10     PC11     PC12     PC13
## Standard deviation     0.01961 0.008564 0.006739 0.004442 0.003361 0.001867
## Proportion of Variance 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000
## Cumulative Proportion  1.00000 1.000000 1.000000 1.000000 1.000000 1.000000
##                            PC14      PC15      PC16      PC17      PC18
## Standard deviation     0.001377 0.0009449 0.0008641 0.0007558 0.0006977
## Proportion of Variance 0.000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.000000 1.0000000 1.0000000 1.0000000 1.0000000
##                             PC19      PC20      PC21      PC22      PC23
## Standard deviation     0.0005884 0.0004628 0.0003897 0.0003341 0.0003123
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
```

```
##                             PC24      PC25     PC26      PC27      PC28
## Standard deviation     0.0002721 0.0002616 0.000211 0.0001954 0.0001857
## Proportion of Variance 0.0000000 0.0000000 0.000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.0000000 1.000000 1.0000000 1.0000000
##                             PC29      PC30      PC31      PC32      PC33
## Standard deviation     0.0001729 0.0001656 0.0001539 0.0001473 0.0001392
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##                             PC34      PC35      PC36     PC37     PC38
## Standard deviation     0.0001339 0.0001269 0.0001082 0.000104 9.98e-05
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.000000 0.00e+00
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.000000 1.00e+00
##                             PC39      PC40      PC41      PC42     PC43
## Standard deviation     9.081e-05 8.668e-05 8.026e-05 7.762e-05 7.36e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.00e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.00e+00
##                             PC44      PC45     PC46      PC47      PC48
## Standard deviation     6.808e-05 6.541e-05 6.44e-05 5.897e-05 5.422e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.00e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.00e+00 1.000e+00 1.000e+00
##                             PC49      PC50      PC51      PC52      PC53
## Standard deviation     5.027e-05 4.893e-05 4.608e-05 4.419e-05 4.037e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC54    PC55     PC56      PC57      PC58      PC59
## Standard deviation     3.854e-05 3.8e-05 3.64e-05 3.497e-05 3.443e-05 3.264e-05
## Proportion of Variance 0.000e+00 0.0e+00 0.00e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.0e+00 1.00e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC60     PC61      PC62      PC63      PC64
## Standard deviation     3.104e-05 3.04e-05 2.959e-05 2.844e-05 2.699e-05
## Proportion of Variance 0.000e+00 0.00e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.00e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC65      PC66      PC67      PC68      PC69
## Standard deviation     2.586e-05 2.388e-05 2.364e-05 2.284e-05 2.173e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC70      PC71     PC72      PC73      PC74
## Standard deviation     2.058e-05 1.997e-05 1.93e-05 1.854e-05 1.807e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.00e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.00e+00 1.000e+00 1.000e+00
##                             PC75      PC76      PC77      PC78      PC79
## Standard deviation     1.728e-05 1.693e-05 1.612e-05 1.569e-05 1.516e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC80      PC81      PC82      PC83      PC84
## Standard deviation     1.445e-05 1.408e-05 1.356e-05 1.275e-05 1.224e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC85     PC86      PC87      PC88      PC89
## Standard deviation     1.178e-05 1.09e-05 1.045e-05 1.009e-05 9.396e-06
## Proportion of Variance 0.000e+00 0.00e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.00e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC90     PC91      PC92     PC93      PC94
## Standard deviation     8.728e-06 8.27e-06 7.613e-06 6.83e-06 6.383e-06
```

```
## Proportion of Variance 0.000e+00 0.00e+00 0.000e+00 0.00e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.00e+00 1.000e+00 1.00e+00 1.000e+00
##                             PC95      PC96      PC97      PC98      PC99
## Standard deviation     5.946e-06 5.478e-06 4.826e-06 4.521e-06 4.164e-06
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC100
## Standard deviation     4.122e-06
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

The first principal explains 98.63% of the variance, thus the data is effectively unidimensional.

6.1 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)?

```
set.seed(111)

absorppca <- pc$x[,1:2]

train <- createDataPartition(endpoints[,3], p=.80, list=F)

predicttrain <- as.data.frame(absorppca[train,])
predicttest <- as.data.frame(absorppca[-train,])
outcometrain <- endpoints[train, 3]
outcometest <- endpoints[-train, 3]
```

Use the mean centered and scaled first two principal components as the transformed predictors.

Train models on 80% of the data. Outcome we are predicting is percentage of protein.

Train a linear regression model using 10-fold cross-validation

```
set.seed(111)
lm <- train(x=predicttrain,
            y=outcometrain,
            method='lm',
            trControl=trainControl(method="cv", number=10))

lm
```

```
## Linear Regression
##
## 174 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 157, 157, 155, 158, 156, 156, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   2.685842  0.2251451  2.185319
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Train a partial least squares model using 10-fold cross-validation

```
set.seed(111)
pls <- train(x=predicttrain,
             y=outcometrain,
             method='pls',
             trControl=trainControl(method="cv", number=10),
             tuneLength=10)

pls
```

```
## Partial Least Squares
##
## 174 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 157, 157, 155, 158, 156, 156, ...
## Resampling results:
##
##   RMSE      Rsquared    MAE
##   2.921534  0.06913596  2.51244
##
## Tuning parameter 'ncomp' was held constant at a value of 1
```

Train a lasso regression model using 10-fold cross-validation

```
set.seed(111)
lasso <- train(x=predicttrain,
               y=outcometrain,
               method='lasso',
               trControl=trainControl(method="cv", number=10),
               tuneLength=10)

lasso
```

```
## The lasso
##
## 174 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 157, 157, 155, 158, 156, 156, ...
## Resampling results across tuning parameters:
##
##   fraction   RMSE      Rsquared   MAE
##   0.1000000  2.928445  0.1806211  2.513207
##   0.1888889  2.872975  0.1806211  2.452761
##   0.2777778  2.832459  0.1784288  2.404256
##   0.3666667  2.803259  0.1853025  2.370335
##   0.4555556  2.775826  0.1931346  2.338611
```

```
##   0.5444444  2.747956  0.2021631  2.307922
##   0.6333333  2.724910  0.2103343  2.277800
##   0.7222222  2.707100  0.2164868  2.247677
##   0.8111111  2.694605  0.2206637  2.217725
##   0.9000000  2.687466  0.2233417  2.194344
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 0.9.
```

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

LASSO regularization and linear regression had the lowest cross-validation error (RMSE = 2.69) which were both superior to partial least squares (RMSE = 2.92)

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

I would use linear regression as it's the more parsimonious model and has equivalent performance to the LASSO regression.

6.2 a)

```
library(AppliedPredictiveModeling)
```

6.2 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?

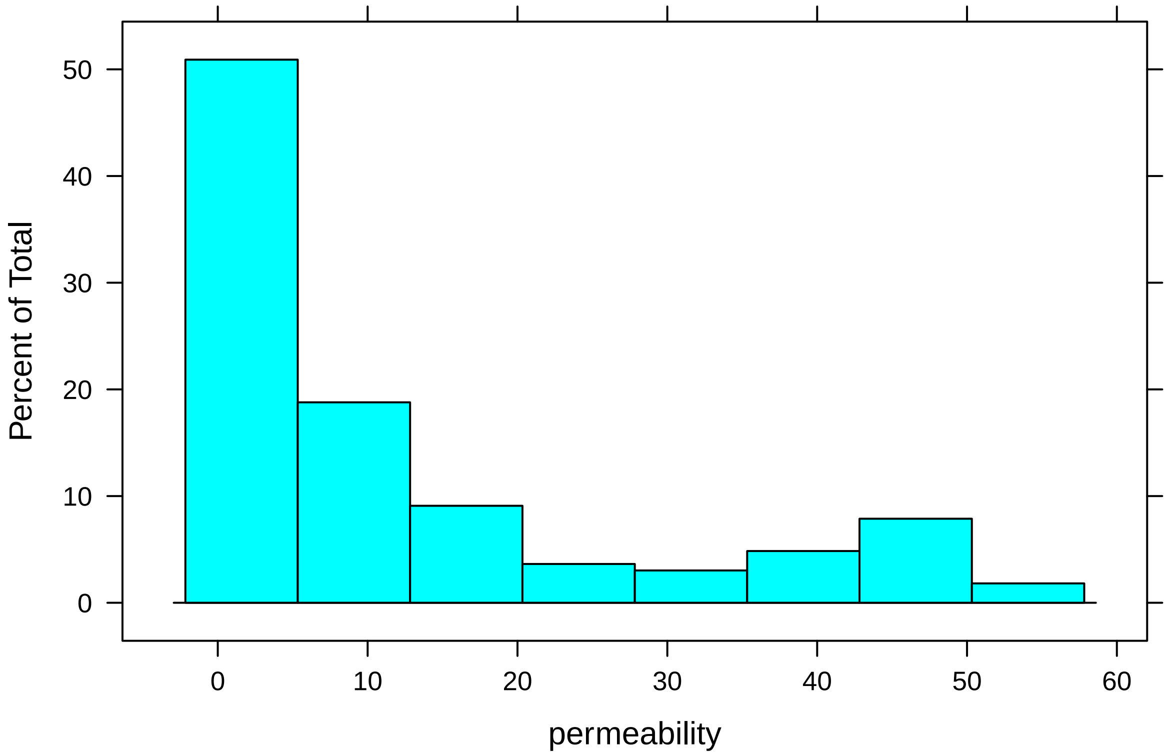
```
data("permeability")
fingerprints <- fingerprints[,-nearZeroVar(fingerprints)]
ncol(fingerprints)
```

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

719 predictors have low frequencies, 388 are left for modeling.

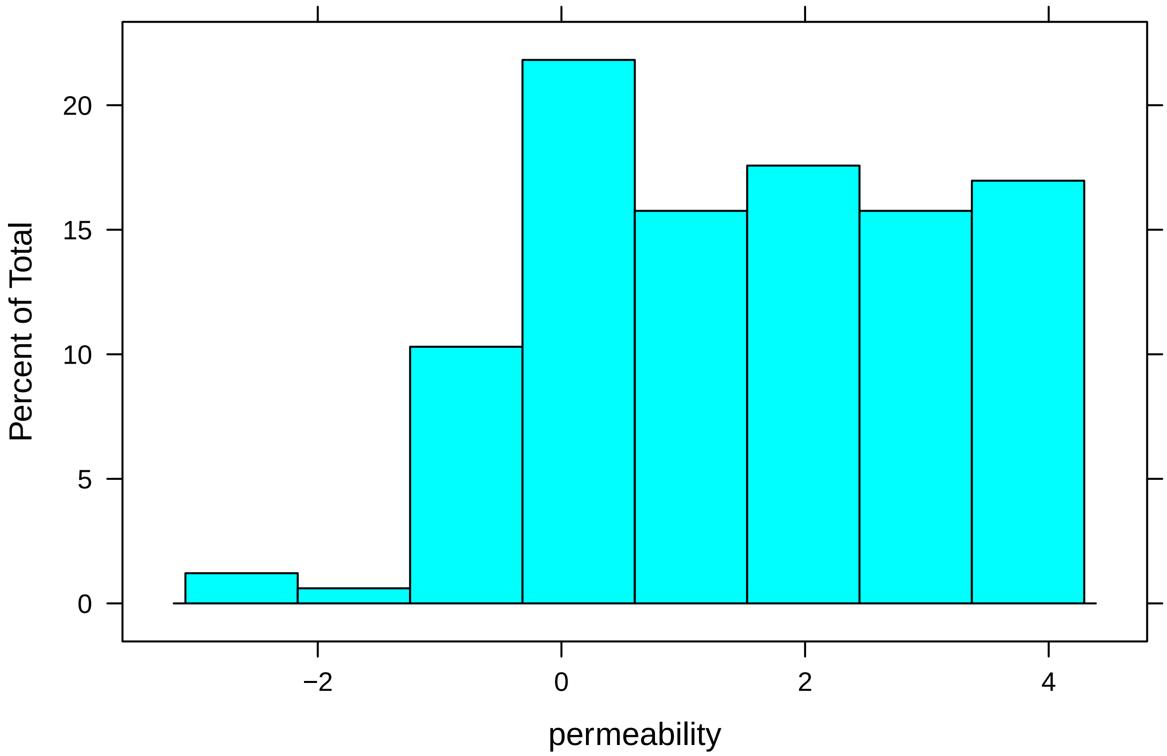
6.2 c) Split the data into a training and a test set, pre-process the data, and tune a PLS model. How many latent variables are optimal and what is the corresponding resampled estimate of R2?

```
histogram(permeability)
```



Heavily skewed, do log transformation

```
permeability <- log(permeability)
histogram(permeability)
```



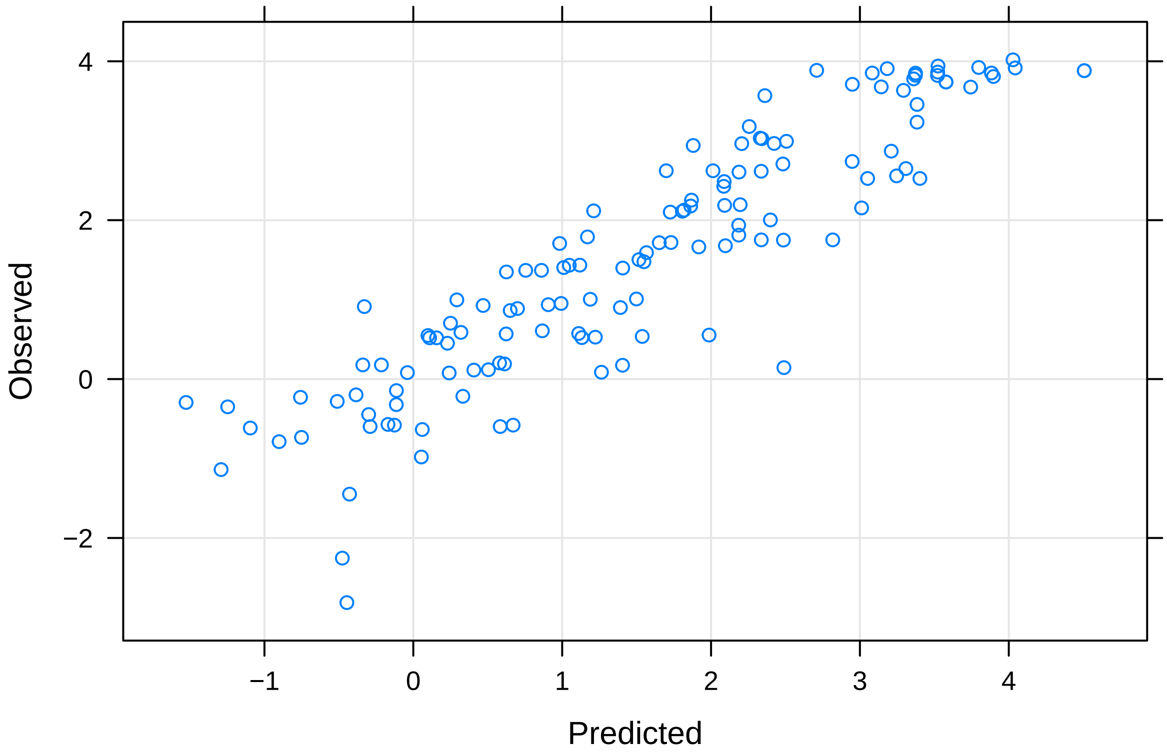

```
set.seed(111)

train <- createDataPartition(permeability,
 p = 0.80, list = F)

predicttrain <- fingerprints[train,]
predicttest <- fingerprints[-train,]

outcometrain <- permeability[train,]
outcometest <- permeability[-train,]
```

```
set.seed(111)

pls <- train(x = predicttrain, y = outcometrain,
 method = "pls",
 preProcess = c("center","scale"),
 tuneLength=20,
 trControl = trainControl(method="cv", number=10))

pls
```

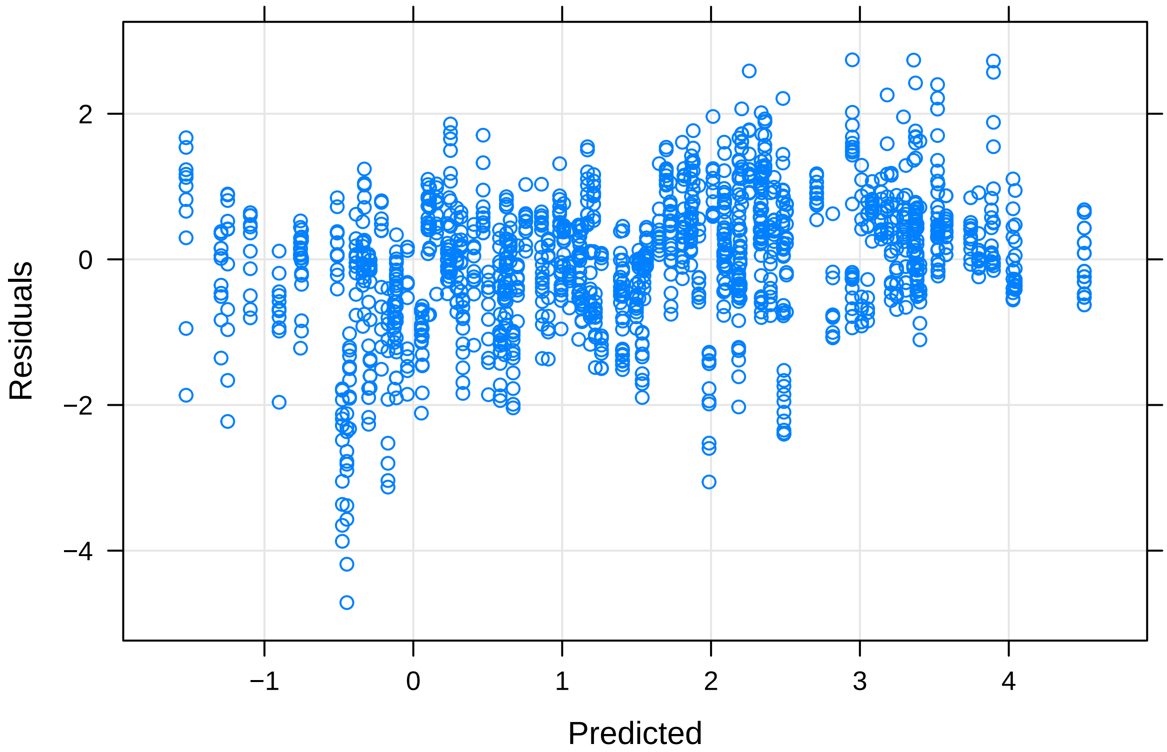
```
## Partial Least Squares
##
## 133 samples
## 388 predictors
```

```
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 119, 120, 120, 120, 119, 121, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    1     1.361817  0.2859507  1.1221819
##    2     1.273867  0.3624227  0.9937406
##    3     1.228694  0.4130219  0.9579445
##    4     1.235617  0.4070234  0.9946070
##    5     1.225362  0.4173404  0.9787958
##    6     1.233122  0.4225346  0.9900852
##    7     1.184144  0.4554283  0.9327960
##    8     1.159239  0.4882521  0.9240490
##    9     1.164785  0.4858744  0.9167945
##   10     1.153540  0.5035400  0.9234391
##   11     1.152530  0.5089720  0.9420248
##   12     1.162512  0.5096490  0.9333737
##   13     1.196117  0.4893859  0.9509678
##   14     1.215972  0.4817467  0.9581004
##   15     1.223155  0.4786315  0.9688465
##   16     1.220519  0.4829882  0.9691338
##   17     1.226320  0.4829575  0.9800187
##   18     1.229828  0.4872436  0.9786228
##   19     1.224106  0.4958762  0.9725801
##   20     1.216429  0.4990197  0.9742153
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 11.
```

```
xyplot(outcometrain ~ predict(pls),
  type = c("p", "g"),
  xlab = "Predicted", ylab = "Observed")
```




```
xyplot(resid(pls) ~ predict(pls),
  type = c("p", "g"),
  xlab = "Predicted", ylab = "Residuals")
```



10 latent variables achieve the lowest RMSE and highest R2.

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

```
plspred <- predict(pls, predicttest)

plsvalues  <- data.frame(obs = outcometest, pred = plspred)

defaultSummary(plsvalues)
```

```
##      RMSE  Rsquared       MAE
## 1.0760388 0.4803441 0.8267802
```

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

Train a ridge regression using 5-fold cross-validation

```
set.seed(111)

# ridge <- train(x = predicttrain, y = outcometrain,
#  method = "ridge",
#  preProcess = c("center","scale"),
#  tuneGrid = data.frame(.lambda = seq(0, .1, length = 15)),
#  trControl = trainControl(method="cv", number=5))
#
# save(ridge, file='ridge.RData')
load('ridge.RData')
```

```r
ridge
```

```
## Ridge Regression
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 106, 107, 106, 107, 106
## Resampling results across tuning parameters:
##
##   lambda       RMSE        Rsquared   MAE
##   0.000000000    1.319972  0.4085111    1.009060
##   0.007142857    3.350625  0.2903132    2.446185
##   0.014285714  774.120359  0.3699128  479.023479
##   0.021428571    1.352135  0.4515592    1.050329
##   0.028571429    6.725144  0.3334906    4.381025
##   0.035714286   34.057396  0.4208521   23.866579
##   0.042857143    1.317463  0.4676598    1.022785
##   0.050000000    1.492769  0.4386049    1.180946
##   0.057142857    1.312524  0.4701563    1.018393
##   0.064285714    1.306514  0.4712884    1.014732
##   0.071428571    1.302865  0.4730895    1.012398
##   0.078571429    1.293484  0.4849745    1.005130
##   0.085714286    7.775697  0.3626397    5.955507
##   0.092857143    1.291983  0.4835143    1.008053
##   0.100000000    1.293239  0.4831173    1.011496
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 0.09285714.
```

RMSE = 1.29, R2 = 0.48

```r
ridgepred <- predict(ridge, predicttest)

ridgevalues  <- data.frame(obs = outcometest, pred = ridgepred)

defaultSummary(ridgevalues)
```

```
##      RMSE  Rsquared       MAE
## 1.1139603 0.4564813 0.8882892
```

Ridge has worse R2 than PLS

Train PCR model with 10-fold cross-validation

```r
set.seed(111)
pcr <- train(x=predicttrain,
             y=outcometrain,
             preProcess = c("center","scale"),
             method='pcr',
```

```
             trControl=trainControl(method="cv", number=10),
             tuneLength=10)

pcr
```

```
## Principal Component Analysis
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 119, 120, 120, 120, 119, 121, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    1     1.536723  0.1411821  1.309405
##    2     1.544169  0.1633703  1.319701
##    3     1.406755  0.2551876  1.183531
##    4     1.385085  0.2567968  1.145100
##    5     1.376367  0.2611917  1.120599
##    6     1.365820  0.2795949  1.116338
##    7     1.372591  0.2698209  1.121214
##    8     1.291142  0.3423978  1.010132
##    9     1.290559  0.3482430  1.013489
##   10     1.263537  0.3780696  1.008162
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 10.
```

RMSE = 1.28, R2 = 0.36

```
pcrpred <- predict(pcr, predicttest)

pcrvalues  <- data.frame(obs = outcometest, pred = pcrpred)

defaultSummary(pcrvalues)
```

```
##      RMSE  Rsquared       MAE
## 1.0015472 0.4834962 0.7641258
```

PCR has worse prediction than ridge regression and PLS

Train elastic net model with 5-fold cross-validation

```
set.seed(111)
enet <- train(x=predicttrain,
             y=outcometrain,
             preProcess = c("center","scale"),
             method='enet',
             tuneGrid= expand.grid(.lambda = c(0, 0.01, .1), .fraction = seq(.05, 1, length = 10)),
             trControl=trainControl(method="cv", number=5))

enet
```

```
## Elasticnet
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 106, 107, 106, 107, 106
## Resampling results across tuning parameters:
##
##   lambda  fraction   RMSE      Rsquared   MAE
##   0.00    0.0500000  1.269228  0.4775161  1.0210681
##   0.00    0.1555556  1.078746  0.5618466  0.8299725
##   0.00    0.2611111  1.076573  0.5543305  0.8283616
##   0.00    0.3666667  1.108182  0.5258323  0.8373911
##   0.00    0.4722222  1.139142  0.5042508  0.8683956
##   0.00    0.5777778  1.186092  0.4763381  0.9034614
##   0.00    0.6833333  1.224020  0.4527578  0.9282800
##   0.00    0.7888889  1.255521  0.4359204  0.9565751
##   0.00    0.8944444  1.286535  0.4226728  0.9837701
##   0.00    1.0000000  1.319972  0.4085111  1.0090603
##   0.01    0.0500000  1.128791  0.5314639  0.8768117
##   0.01    0.1555556  1.105230  0.5339432  0.8437616
##   0.01    0.2611111  1.172913  0.4929119  0.9111233
##   0.01    0.3666667  1.273250  0.4364608  0.9805456
##   0.01    0.4722222  1.342474  0.4074904  1.0241591
##   0.01    0.5777778  1.378135  0.4076533  1.0486581
##   0.01    0.6833333  1.390637  0.4161963  1.0664647
##   0.01    0.7888889  1.402260  0.4190851  1.0831981
##   0.01    0.8944444  1.400607  0.4245076  1.0934731
##   0.01    1.0000000  1.404003  0.4258330  1.0983810
##   0.10    0.0500000  1.216470  0.5080327  0.9651157
##   0.10    0.1555556  1.084067  0.5495152  0.8231967
##   0.10    0.2611111  1.096769  0.5466965  0.8359111
##   0.10    0.3666667  1.132554  0.5249827  0.8761318
##   0.10    0.4722222  1.172898  0.5059062  0.9169705
##   0.10    0.5777778  1.207595  0.4917331  0.9436426
##   0.10    0.6833333  1.229600  0.4882426  0.9593976
##   0.10    0.7888889  1.255547  0.4836060  0.9785231
##   0.10    0.8944444  1.277108  0.4821957  0.9974370
##   0.10    1.0000000  1.293239  0.4831173  1.0114959
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were fraction = 0.2611111 and lambda = 0.
```

RMSE = 1.08, R2 = 0.55

```r
enetpred <- predict(enet, predicttest)

enetvalues  <- data.frame(obs = outcometest, pred = enetpred)

defaultSummary(enetvalues)
```

```
##      RMSE  Rsquared       MAE
```

```
## 1.4450554 0.2421683 1.1239332
```

enet did not perform well on the testset? Maybe I did something wrong

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

PLS maybe as it had good performance

7.4. Return to the permeability problem outlined in Exercise 6.2. Train several nonlinear regression models and evaluate the resampling and test set performance.

Train a neural net using leave group out cross-validation

```
# set.seed(111)
# nnet <- train(predicttrain, outcometrain,
#                    method = "nnet",
#                    tuneGrid = expand.grid(size = c(1,3,5,7), decay = c(0, .01, .1)),
#                    trControl = trainControl(method="LGOCV"),
#                    preProc = c("center", "scale"),
#                    linout = TRUE,
#                    trace = FALSE,
#                    MaxNWts = 10 * (ncol(predicttrain) + 1) + 10 + 1,
#                    maxit = 500)
#
#
# save(nnet, file='nnet.RData')
# load('nnet.RData')
#
# nnet
# predict(nnet, predicttest)
```

Computer not fast enough to run

Train a MARS model

```
# set.seed(111)
# mars <- train(predicttrain, outcometrain,
#               method = "earth",
#               trControl = trainControl(method="LGOCV"),
#               preProc = c("center", "scale"),
#               tuneGrid = expand.grid(degree=1,nprune=2:30))
# save(mars, file='mars.RData')
load('mars.RData')

mars
```

```
## Multivariate Adaptive Regression Spline
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
```

```
##
##   nprune  RMSE      Rsquared   MAE
##    2      1.246503  0.3876209  0.9586671
##    3      1.192706  0.4364056  0.9069172
##    4      1.118214  0.5071330  0.8526778
##    5      1.150427  0.4827117  0.8872623
##    6      1.175472  0.4720425  0.9055658
##    7      1.188159  0.4671005  0.9194881
##    8      1.207163  0.4575526  0.9343908
##    9      1.215845  0.4513641  0.9437186
##   10      1.240098  0.4515237  0.9548940
##   11      1.233967  0.4564379  0.9531122
##   12      1.222630  0.4636963  0.9419995
##   13      1.229614  0.4610203  0.9497313
##   14      1.248229  0.4473307  0.9610486
##   15      1.287027  0.4264664  0.9903963
##   16      1.295644  0.4222583  0.9977627
##   17      1.301058  0.4208008  1.0027932
##   18      1.299919  0.4214132  1.0016310
##   19      1.330813  0.4228112  1.0186291
##   20      1.332369  0.4224641  1.0200179
##   21      1.332369  0.4224641  1.0200179
##   22      1.332369  0.4224641  1.0200179
##   23      1.332369  0.4224641  1.0200179
##   24      1.332369  0.4224641  1.0200179
##   25      1.332369  0.4224641  1.0200179
##   26      1.332369  0.4224641  1.0200179
##   27      1.332369  0.4224641  1.0200179
##   28      1.332369  0.4224641  1.0200179
##   29      1.332369  0.4224641  1.0200179
##   30      1.332369  0.4224641  1.0200179
##
## Tuning parameter 'degree' was held constant at a value of 1
## 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.
```

```
marspred <- predict(mars, predicttest)
marsvalues <- data.frame(obs=outcometest, pred=marspred)
# defaultSummary(marsvalues)
# Error in `[.data.frame`(data, , "pred") : undefined columns selected
```

RMSE = 1.1, R2 = 0.51

Train support vector machine model

```
set.seed(111)
# svm <- train(predicttrain, outcometrain,
#               method = "svmRadial",
#               trControl = trainControl(method="LGOCV"),
#               preProc = c("center", "scale"),
#               tuneLength=10)
# save(svm, file='svm.RData')
load('svm.RData')
```

```
svm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   C       RMSE      Rsquared   MAE
##     0.25  1.334733  0.3137700  1.0521023
##     0.50  1.291651  0.3404473  0.9737115
##     1.00  1.269121  0.3637962  0.9443296
##     2.00  1.249728  0.3856859  0.9244178
##     4.00  1.207571  0.4291871  0.8823790
##     8.00  1.198391  0.4496673  0.8800758
##    16.00  1.220817  0.4377273  0.8947350
##    32.00  1.214212  0.4442266  0.8984117
##    64.00  1.222711  0.4388214  0.9063286
##   128.00  1.247967  0.4234742  0.9267731
##
## Tuning parameter 'sigma' was held constant at a value of 0.001884705
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.001884705 and C = 8.
```

```
svmpred <- predict(svm, predicttest)
svmvalues <- data.frame(obs=outcometest, pred=svmpred)
defaultSummary(svmvalues)
```

```
##      RMSE  Rsquared       MAE
## 1.0244377 0.4977535 0.8564069
```

RMSE = 1.20, R2 = 0.45

```
# set.seed(111)
# knn <- train(predicttrain, outcometrain,
#              method = "knn",
#              trControl = trainControl(method="LGOCV"),
#              preProc = c("center", "scale"),
#              tuneGrid = data.frame(k=1:20))
#
# save(knn, file='knn.RData')
load('knn.RData')

knn
```

```
## k-Nearest Neighbors
##
```

```
## 133 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
## Summary of sample sizes: 101, 101, 101, 101, 101, 101, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    1  1.627614  0.2568991  1.123644
##    2  1.424850  0.2958606  1.042506
##    3  1.404848  0.2769894  1.025230
##    4  1.401865  0.2613045  1.026418
##    5  1.415073  0.2467192  1.038263
##    6  1.418134  0.2390303  1.053714
##    7  1.411089  0.2386884  1.058866
##    8  1.404160  0.2395313  1.070962
##    9  1.407892  0.2337906  1.084970
##   10  1.397531  0.2407304  1.085844
##   11  1.400823  0.2345925  1.101042
##   12  1.396121  0.2386286  1.104711
##   13  1.396631  0.2372868  1.113591
##   14  1.394906  0.2362277  1.117825
##   15  1.389499  0.2395826  1.115704
##   16  1.390119  0.2404707  1.118364
##   17  1.396254  0.2346873  1.128635
##   18  1.398081  0.2339279  1.135098
##   19  1.402332  0.2292371  1.143638
##   20  1.404015  0.2287953  1.149343
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 15.
```

```r
knnpred <- predict(knn, predicttest)
knnvalues <- data.frame(obs=outcometest, pred=knnpred)
defaultSummary(knnvalues)
```

```
##      RMSE  Rsquared       MAE
## 1.1061154 0.3472030 0.9275839
```

RMSE = 1.39, R2 = 0.24

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

The MARS model had the best fit to the training data, but for some reason there was an error in calculating the prediction fit on the test data. SVM had greater prediction fit than KNN.

b) Do any of the nonlinear models outperform the optimal linear model you previously developed in Exercise 6.2? If so, what might this tell you about the underlying relationship between the predictors and the response

SVM outperformed PLS, so the relationship between predictors and response may be better characterized as nonlinear

c) Would you recommend any of the models you have developed to replace the permeability laboratory experiment?

Perhaps SVM or MARS