Santanu_Mukherjee_zes254_HW2

Santanu Mukherjee, zes254

07/03/2022

R Markdown

Chapter 6 - E-Book - Applied Predictive Modelling - Chapter 6 - Exercises pages 137 - 138:

Q 6.1

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. A sample of these frequency profiles is displayed in Fig. 6.20. 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

6.1 a

Start R and use these commands to load the data

Answer (6.1 a)

```
library(caret)
data(tecator)
# use ?tecator to see more details
```

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.

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?

Answer (6.1 b)

The function prcomp is used for PCA.

```
pca.b = prcomp(absorp, center = TRUE, scale = TRUE )

# Determining percent of variance associated with each component

pct.var = pca.b$sdev^2/sum(pca.b$sdev^2) * 100
head(pct.var)
```

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

The pct. var data shows that the first components accounts for almost 99% of the information in the data. So, reviewing this analysis, we can say that the true real dimensionality is much lower than the total number of predictors. But this is considering the linear combination of data. If we try to use non-linear combinations of the predictors, we might get a different result.

6.1 c

Split the data into a training and a test set the response of the percentage of protein, 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)?

Answer (6.1 c)

So, there are 3 endpoints and based on the question, we will model the percentage of protein (third column). We will do the split of the dataset into 80% (train) and 20% (test), by using createDataPartition function. We will also perform 10-fold Cross validation (CV) with 5 repeats to tune the models.

```
set.seed(10)
train.part = createDataPartition(endpoints[,3], p = 0.8, list=FALSE)
absorp = data.frame(absorp)

absorp.train = absorp[train.part,]
absorp.test = absorp[-train.part,]
protein.train = endpoints[train.part,3]
protein.test = endpoints[-train.part,3]

cntrl = trainControl(method = "repeatedcv", repeats = 5)
```

```
# Build various models and then compare performance:

# Simple Linear Model
set.seed(15)
lm.model = train( absorp.train, protein.train, method="lm", preProcess=c("center","scale"), trControl=trainControl(method="r
epeatedcv",repeats=5) )
lm.model
```

```
## Linear Regression
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 157, 157, 156, 156, 157, 157, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.214111  0.8403278  0.7567567
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
# For Robust Linear Model (rlm) we cannot have a singular predictor covariance matrix thus we preprocess with PCA:
#
set.seed(0)
rlm.model = train( absorp.train, protein.train, method="rlm", preProcess=c("pca"), trControl=trainControl(method="repeatedc
v",repeats=5) )
rlm.model
```

```
## Robust Linear Model
##
## 174 samples
## 100 predictors
##
## Pre-processing: principal component signal extraction (100), centered
##  (100), scaled (100)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 156, 157, 155, 157, 157, 158, ...
## Resampling results across tuning parameters:
##
##   intercept  psi           RMSE       Rsquared   MAE
##   FALSE      psi.huber     17.874222  0.2589858  17.679098
##   FALSE      psi.hampel    17.874222  0.2589858  17.679098
##   FALSE      psi.bisquare  17.874468  0.2589351  17.679312
##    TRUE      psi.huber      2.646350  0.2586091   2.123615
##    TRUE      psi.hampel     2.639445  0.2590020   2.146582
##    TRUE      psi.bisquare   2.644695  0.2589679   2.122705
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were intercept = TRUE and psi = psi.hampel.
```

```r
# Ridge regression model
#
ridgeGrid = data.frame(.lambda=seq(0,1,length=20))
set.seed(0)
ridge.model = train( absorp.train, protein.train, method="ridge",
                         # fit the model over many penalty values
                         tuneGrid = ridgeGrid,
                         preProcess=c("center","scale"), trControl=trainControl(method="repeatedcv",repeats=5) )

ridge.model
```

```
## Ridge Regression
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 156, 157, 155, 157, 157, 158, ...
## Resampling results across tuning parameters:
##
##   lambda      RMSE      Rsquared   MAE
##   0.00000000  1.193874  0.8476045  0.7342679
##   0.05263158  1.636625  0.7208033  1.2419305
##   0.10526316  1.757456  0.6856649  1.3560860
##   0.15789474  1.841890  0.6585549  1.4422808
##   0.21052632  1.906368  0.6354663  1.5066399
##   0.26315789  1.957833  0.6152083  1.5549893
##   0.31578947  2.000302  0.5971100  1.5937334
##   0.36842105  2.036315  0.5807153  1.6257207
##   0.42105263  2.067569  0.5656934  1.6523351
##   0.47368421  2.095236  0.5517969  1.6750961
##   0.52631579  2.120158  0.5388370  1.6954747
##   0.57894737  2.142950  0.5266674  1.7147399
##   0.63157895  2.164077  0.5151738  1.7323326
##   0.68421053  2.183893  0.5042655  1.7486696
##   0.73684211  2.202673  0.4938707  1.7638205
##   0.78947368  2.220635  0.4839314  1.7780568
##   0.84210526  2.237953  0.4744009  1.7918705
##   0.89473684  2.254765  0.4652412  1.8054722
##   0.94736842  2.271187  0.4564213  1.8189786
##   1.00000000  2.287311  0.4479154  1.8323720
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 0.
```

```r
# Elastic net model
#
enetGrid = expand.grid(.lambda=seq(0,1,length=20), .fraction=seq(0.05, 1.0, length=20))
set.seed(0)
enet.model = train( absorp.train, protein.train, method="enet",
                    # fit the model over many penalty values
                    tuneGrid = enetGrid,
                    preProcess=c("center","scale"), trControl=trainControl(method="repeatedcv",repeats=5) )


enet.model
```

```
## Elasticnet
##
## 174 samples
## 100 predictors
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 156, 157, 155, 157, 157, 158, ...
## Resampling results across tuning parameters:
##
##   lambda      fraction  RMSE       Rsquared   MAE
##   0.00000000  0.05      0.7762276  0.9298649  0.5308600
##   0.00000000  0.10      0.8186195  0.9209959  0.5370261
##   0.00000000  0.15      0.8529118  0.9175913  0.5555308
##   0.00000000  0.20      0.8631860  0.9175482  0.5587314
##   0.00000000  0.25      0.8675890  0.9172100  0.5589071
##   0.00000000  0.30      0.8646685  0.9182009  0.5581328
##   0.00000000  0.35      0.8737529  0.9167591  0.5655718
##   0.00000000  0.40      0.8925406  0.9123552  0.5767281
##   0.00000000  0.45      0.9141198  0.9074089  0.5881487
##   0.00000000  0.50      0.9407166  0.9015415  0.6025995
##   0.00000000  0.55      0.9717213  0.8953855  0.6198332
##   0.00000000  0.60      1.0005139  0.8895083  0.6346006
##   0.00000000  0.65      1.0285492  0.8830115  0.6476897
##   0.00000000  0.70      1.0539835  0.8774505  0.6603950
##   0.00000000  0.75      1.0807572  0.8717840  0.6743307
##   0.00000000  0.80      1.1042389  0.8668441  0.6868856
##   0.00000000  0.85      1.1271631  0.8619416  0.6990853
##   0.00000000  0.90      1.1504322  0.8568343  0.7113099
##   0.00000000  0.95      1.1734603  0.8518945  0.7233594
##   0.00000000  1.00      1.1938739  0.8476045  0.7342679
##   0.05263158  0.05      2.8424192  0.1789051  2.4262706
##   0.05263158  0.10      2.7288617  0.2932349  2.3272874
##   0.05263158  0.15      2.6180865  0.4008755  2.2296753
##   0.05263158  0.20      2.5109118  0.4878265  2.1360977
##   0.05263158  0.25      2.4077837  0.5523688  2.0442639
##   0.05263158  0.30      2.3095922  0.5980329  1.9548461
##   0.05263158  0.35      2.2170070  0.6303191  1.8688886
##   0.05263158  0.40      2.1296001  0.6535016  1.7854741
##   0.05263158  0.45      2.0479268  0.6701920  1.7050947
##   0.05263158  0.50      1.9738242  0.6823600  1.6281307
##   0.05263158  0.55      1.9068228  0.6915382  1.5549507
##   0.05263158  0.60      1.8475708  0.6985281  1.4870992
##   0.05263158  0.65      1.7957938  0.7039866  1.4281369
##   0.05263158  0.70      1.7531474  0.7083627  1.3795142
##   0.05263158  0.75      1.7184225  0.7120175  1.3396816
##   0.05263158  0.80      1.6910890  0.7147917  1.3060550
##   0.05263158  0.85      1.6704918  0.7170113  1.2807446
##   0.05263158  0.90      1.6535574  0.7190833  1.2606577
##   0.05263158  0.95      1.6428839  0.7200021  1.2483446
##   0.05263158  1.00      1.6366250  0.7208033  1.2419305
##   0.10526316  0.05      2.8605970  0.1582484  2.4376115
##   0.10526316  0.10      2.7634306  0.2490712  2.3523990
##   0.10526316  0.15      2.6686723  0.3399186  2.2684214
##   0.10526316  0.20      2.5770638  0.4198039  2.1883321
##   0.10526316  0.25      2.4894219  0.4836384  2.1110066
##   0.10526316  0.30      2.4055848  0.5327742  2.0353393
##   0.10526316  0.35      2.3251727  0.5698881  1.9611665
##   0.10526316  0.40      2.2494533  0.5974653  1.8897894
##   0.10526316  0.45      2.1785901  0.6183208  1.8217289
##   0.10526316  0.50      2.1126784  0.6341758  1.7568468
##   0.10526316  0.55      2.0515034  0.6464933  1.6940402
##   0.10526316  0.60      1.9955242  0.6560519  1.6340399
##   0.10526316  0.65      1.9453893  0.6637205  1.5783557
##   0.10526316  0.70      1.9009760  0.6698787  1.5291424
##   0.10526316  0.75      1.8617779  0.6750366  1.4854183
##   0.10526316  0.80      1.8293348  0.6788234  1.4470014
##   0.10526316  0.85      1.8036486  0.6816466  1.4157961
##   0.10526316  0.90      1.7823648  0.6839806  1.3891229
##   0.10526316  0.95      1.7668914  0.6852539  1.3689079
##   0.10526316  1.00      1.7574564  0.6856649  1.3560860
##   0.15789474  0.05      2.8710755  0.1467199  2.4426483
##   0.15789474  0.10      2.7825850  0.2239234  2.3645429
##   0.15789474  0.15      2.6967618  0.3030756  2.2882443
##   0.15789474  0.20      2.6140742  0.3755150  2.2158106
##   0.15789474  0.25      2.5349112  0.4367513  2.1462234
##   0.15789474  0.30      2.4586952  0.4862936  2.0776229
##   0.15789474  0.35      2.3861194  0.5248169  2.0107443
##   0.15789474  0.40      2.3176363  0.5546155  1.9463514
##   0.15789474  0.45      2.2530959  0.5777484  1.8846457
##   0.15789474  0.50      2.1922334  0.5960170  1.8258143
##   0.15789474  0.55      2.1351667  0.6104221  1.7689660
##   0.15789474  0.60      2.0828569  0.6219513  1.7148269
##   0.15789474  0.65      2.0350193  0.6312717  1.6632935
##   0.15789474  0.70      1.9917535  0.6389084  1.6148130
##   0.15789474  0.75      1.9529393  0.6451431  1.5718561
##   0.15789474  0.80      1.9201846  0.6498825  1.5357033
##   0.15789474  0.85      1.8931268  0.6534059  1.5053946
##   0.15789474  0.90      1.8705478  0.6562491  1.4792003
##   0.15789474  0.95      1.8534476  0.6578874  1.4578397
##   0.15789474  1.00      1.8418904  0.6585549  1.4422808
##   0.21052632  0.05      2.8787085  0.1389535  2.4451749
##   0.21052632  0.10      2.7961607  0.2065440  2.3719563
##   0.21052632  0.15      2.7163817  0.2767850  2.3008215
##   0.21052632  0.20      2.6398250  0.3427667  2.2338256
##   0.21052632  0.25      2.5661976  0.4008608  2.1690262
##   0.21052632  0.30      2.4954576  0.4493265  2.1053622
##   0.21052632  0.35      2.4284263  0.4881484  2.0436578
##   0.21052632  0.40      2.3646146  0.5192564  1.9837231
##   0.21052632  0.45      2.3041786  0.5440558  1.9262124
##   0.21052632  0.50      2.2468915  0.5639949  1.8714402
##   0.21052632  0.55      2.1933552  0.5799319  1.8190213
##   0.21052632  0.60      2.1438125  0.5929360  1.7685593
##   0.21052632  0.65      2.0982073  0.6035015  1.7201991
##   0.21052632  0.70      2.0564602  0.6122627  1.6742751
##   0.21052632  0.75      2.0189374  0.6193488  1.6317319
##   0.21052632  0.80      1.9866272  0.6249063  1.5971476
##   0.21052632  0.85      1.9594613  0.6290820  1.5681637
##   0.21052632  0.90      1.9366050  0.6324287  1.5429164
##   0.21052632  0.95      1.9189445  0.6344674  1.5222440
##   0.21052632  1.00      1.9063683  0.6354663  1.5066399
##   0.26315789  0.05      2.8848837  0.1332052  2.4463035
##   0.26315789  0.10      2.8068991  0.1934872  2.3768514
##   0.26315789  0.15      2.7316381  0.2566324  2.3096196
##   0.26315789  0.20      2.6595570  0.3171075  2.2466277
##   0.26315789  0.25      2.5901201  0.3718170  2.1852889
##   0.26315789  0.30      2.5236000  0.4186189  2.1253909
##   0.26315789  0.35      2.4603332  0.4573843  2.0672545
##   0.26315789  0.40      2.4000121  0.4891788  2.0108251
##   0.26315789  0.45      2.3426212  0.5151525  1.9566899
##   0.26315789  0.50      2.2881380  0.5362994  1.9047111
##   0.26315789  0.55      2.2373536  0.5534400  1.8553621
##   0.26315789  0.60      2.1900283  0.5675759  1.8076396
##   0.26315789  0.65      2.1464130  0.5791439  1.7620069
##   0.26315789  0.70      2.1061776  0.5887857  1.7185962
##   0.26315789  0.75      2.0699752  0.5966122  1.6783847
##   0.26315789  0.80      2.0383596  0.6028508  1.6435841
##   0.26315789  0.85      2.0114895  0.6076132  1.6153713
##   0.26315789  0.90      1.9887684  0.6114215  1.5911214
##   0.26315789  0.95      1.9709218  0.6138740  1.5708459
##   0.26315789  1.00      1.9578332  0.6152083  1.5549893
##   0.31578947  0.05      2.8903453  0.1287132  2.4466379
##   0.31578947  0.10      2.8159650  0.1832150  2.3801351
##   0.31578947  0.15      2.7442860  0.2405567  2.3160403
##   0.31578947  0.20      2.6756535  0.2962677  2.2561362
##   0.31578947  0.25      2.6095182  0.3476829  2.1975144
##   0.31578947  0.30      2.5462327  0.3927028  2.1404278
##   0.31578947  0.35      2.4858030  0.4310457  2.0851113
##   0.31578947  0.40      2.4281839  0.4631384  2.0317101
##   0.31578947  0.45      2.3732508  0.4898570  1.9802981
##   0.31578947  0.50      2.3211697  0.5118497  1.9306554
##   0.31578947  0.55      2.2725013  0.5299745  1.8832542
##   0.31578947  0.60      2.2270757  0.5449772  1.8377228
##   0.31578947  0.65      2.1850873  0.5573958  1.7944148
##   0.31578947  0.70      2.1462101  0.5677792  1.7532037
##   0.31578947  0.75      2.1112272  0.5762460  1.7155660
##   0.31578947  0.80      2.0803748  0.5830705  1.6815876
##   0.31578947  0.85      2.0539511  0.5883607  1.6526086
##   0.31578947  0.90      2.0315650  0.5925770  1.6291011
##   0.31578947  0.95      2.0137026  0.5954310  1.6094261
##   0.31578947  1.00      2.0003015  0.5971100  1.5937334
##   0.36842105  0.05      2.8950683  0.1250342  2.4463158
##   0.36842105  0.10      2.8240071  0.1748593  2.3823333
##   0.36842105  0.15      2.7552432  0.2273706  2.3208628
##   0.36842105  0.20      2.6893858  0.2789371  2.2634002
##   0.36842105  0.25      2.6258869  0.3273075  2.2069803
##   0.36842105  0.30      2.5651638  0.3705006  2.1522832
##   0.36842105  0.35      2.5070188  0.4081561  2.0995179
##   0.36842105  0.40      2.4515618  0.4402520  2.0486742
##   0.36842105  0.45      2.3986678  0.4673858  1.9997060
##   0.36842105  0.50      2.3485658  0.4900057  1.9520518
##   0.36842105  0.55      2.3017044  0.5088506  1.9062372
##   0.36842105  0.60      2.2578504  0.5245847  1.8624556
##   0.36842105  0.65      2.2172367  0.5377209  1.8210197
##   0.36842105  0.70      2.1796049  0.5487258  1.7818849
##   0.36842105  0.75      2.1456877  0.5577625  1.7463024
##   0.36842105  0.80      2.1156147  0.5650976  1.7137403
##   0.36842105  0.85      2.0897253  0.5708510  1.6847315
##   0.36842105  0.90      2.0677562  0.5754374  1.6604038
##   0.36842105  0.95      2.0499424  0.5786847  1.6412363
##   0.36842105  1.00      2.0363153  0.5807153  1.6257207
##   0.42105263  0.05      2.8980246  0.1219388  2.4450695
##   0.42105263  0.10      2.8313833  0.1679122  2.3841416
##   0.42105263  0.15      2.7650695  0.2163151  2.3250188
##   0.42105263  0.20      2.7014712  0.2642926  2.2695931
##   0.42105263  0.25      2.6401315  0.3098890  2.2151620
##   0.42105263  0.30      2.5814945  0.3512803  2.1624315
##   0.42105263  0.35      2.5252757  0.3880393  2.1117427
##   0.42105263  0.40      2.4716020  0.4199165  2.0632711
##   0.42105263  0.45      2.4203907  0.4472418  2.0160601
##   0.42105263  0.50      2.3719973  0.4702613  1.9701218
##   0.42105263  0.55      2.3266148  0.4896830  1.9259907
##   0.42105263  0.60      2.2841413  0.5059949  1.8833971
##   0.42105263  0.65      2.2447585  0.5197156  1.8434451
##   0.42105263  0.70      2.2082308  0.5312640  1.8060881
##   0.42105263  0.75      2.1752509  0.5408041  1.7721713
##   0.42105263  0.80      2.1459248  0.5485958  1.7407879
##   0.42105263  0.85      2.1205858  0.5547657  1.7124976
##   0.42105263  0.90      2.0990519  0.5596924  1.6877241
##   0.42105263  0.95      2.0813366  0.5633161  1.6677227
##   0.42105263  1.00      2.0675687  0.5656934  1.6523351
##   0.47368421  0.05      2.8988358  0.1194240  2.4428749
##   0.47368421  0.10      2.8383456  0.1620134  2.3860266
##   0.47368421  0.15      2.7740977  0.2069125  2.3287148
##   0.47368421  0.20      2.7123639  0.2517791  2.2750156
##   0.47368421  0.25      2.6528578  0.2948329  2.2222582
##   0.47368421  0.30      2.5959493  0.3344832  2.1712869
##   0.47368421  0.35      2.5413540  0.3702368  2.1226189
##   0.47368421  0.40      2.4892201  0.4016994  2.0759115
##   0.47368421  0.45      2.4394284  0.4290427  2.0301628
##   0.47368421  0.50      2.3924741  0.4523135  1.9858080
##   0.47368421  0.55      2.3483900  0.4721343  1.9429827
##   0.47368421  0.60      2.3071184  0.4889161  1.9015365
##   0.47368421  0.65      2.2688228  0.5031244  1.8626607
##   0.47368421  0.70      2.2333044  0.5151299  1.8268564
##   0.47368421  0.75      2.2011665  0.5251174  1.7941997
##   0.47368421  0.80      2.1725409  0.5333175  1.7638480
##   0.47368421  0.85      2.1477574  0.5398614  1.7364144
##   0.47368421  0.90      2.1266500  0.5451006  1.7122027
##   0.47368421  0.95      2.1090617  0.5490853  1.6912670
##   0.47368421  1.00      2.0952360  0.5517969  1.6750961
##   0.52631579  0.05      2.8976379  0.1177244  2.4406860
##   0.52631579  0.10      2.8450332  0.1569437  2.3881358
##   0.52631579  0.15      2.7825588  0.1988226  2.3324544
##   0.52631579  0.20      2.7224110  0.2409448  2.2798958
##   0.52631579  0.25      2.6644620  0.2817028  2.2284983
##   0.52631579  0.30      2.6090337  0.3196793  2.1790768
##   0.52631579  0.35      2.5558538  0.3543525  2.1321969
##   0.52631579  0.40      2.5050274  0.3852893  2.0867296
##   0.52631579  0.45      2.4564999  0.4124781  2.0423760
##   0.52631579  0.50      2.4107905  0.4358609  1.9993616
##   0.52631579  0.55      2.3678163  0.4559775  1.9575856
##   0.52631579  0.60      2.3276394  0.4731086  1.9174518
##   0.52631579  0.65      2.2903158  0.4877136  1.8795573
##   0.52631579  0.70      2.2556964  0.5001175  1.8451001
##   0.52631579  0.75      2.2243341  0.5104976  1.8134460
##   0.52631579  0.80      2.1963549  0.5190685  1.7840208
##   0.52631579  0.85      2.1721258  0.5259473  1.7573758
##   0.52631579  0.90      2.1514083  0.5314870  1.7337638
##   0.52631579  0.95      2.1339944  0.5357999  1.7128011
##   0.52631579  1.00      2.1201575  0.5388370  1.6954747
##   0.57894737  0.05      2.8960465  0.1166667  2.4395852
##   0.57894737  0.10      2.8515910  0.1525159  2.3901474
##   0.57894737  0.15      2.7906520  0.1917656  2.3359706
##   0.57894737  0.20      2.7318717  0.2314710  2.2842572
##   0.57894737  0.25      2.6752847  0.2701320  2.2340242
##   0.57894737  0.30      2.6211402  0.3065123  2.1861673
##   0.57894737  0.35      2.5692048  0.3400784  2.1406462
##   0.57894737  0.40      2.5195438  0.3703863  2.0963520
##   0.57894737  0.45      2.4721290  0.3973137  2.0531844
##   0.57894737  0.50      2.4275190  0.4206872  2.0113728
##   0.57894737  0.55      2.3855446  0.4409798  1.9707522
##   0.57894737  0.60      2.3463226  0.4583848  1.9318091
##   0.57894737  0.65      2.3098695  0.4733160  1.8950662
##   0.57894737  0.70      2.2760754  0.4860575  1.8614691
##   0.57894737  0.75      2.2454343  0.4967782  1.8305219
##   0.57894737  0.80      2.2180535  0.5056849  1.8018175
##   0.57894737  0.85      2.1943481  0.5128711  1.7759044
##   0.57894737  0.90      2.1740033  0.5186963  1.7527988
##   0.57894737  0.95      2.1567828  0.5233125  1.7321617
##   0.57894737  1.00      2.1429501  0.5266674  1.7147399
##   0.63157895  0.05      2.8944538  0.1159962  2.4393427
##   0.63157895  0.10      2.8581007  0.1486121  2.3920942
##   0.63157895  0.15      2.7985068  0.1855565  2.3391287
##   0.63157895  0.20      2.7409407  0.2230967  2.2884203
##   0.63157895  0.25      2.6855739  0.2598369  2.2391814
##   0.63157895  0.30      2.6325435  0.2947181  2.1927533
##   0.63157895  0.35      2.5817080  0.3271770  2.1482731
##   0.63157895  0.40      2.5330919  0.3567904  2.1051746
##   0.63157895  0.45      2.4866908  0.3833535  2.0630702
##   0.63157895  0.50      2.4430510  0.4066322  2.0222001
##   0.63157895  0.55      2.4019828  0.4270021  1.9826867
##   0.63157895  0.60      2.3636324  0.4445945  1.9447539
##   0.63157895  0.65      2.3279604  0.4597891  1.9091230
##   0.63157895  0.70      2.2949249  0.4728140  1.8764464
##   0.63157895  0.75      2.2649582  0.4838287  1.8459161
##   0.63157895  0.80      2.2381319  0.4930422  1.8177170
##   0.63157895  0.85      2.2149169  0.5005092  1.7924992
##   0.63157895  0.90      2.1949135  0.5066109  1.7698563
##   0.63157895  0.95      2.1779022  0.5115087  1.7497376
##   0.63157895  1.00      2.1640771  0.5151738  1.7323326
##   0.68421053  0.05      2.8932958  0.1156301  2.4398447
##   0.68421053  0.10      2.8646285  0.1451377  2.3939035
##   0.68421053  0.15      2.8062226  0.1800446  2.3419376
##   0.68421053  0.20      2.7497390  0.2156396  2.2924370
##   0.68421053  0.25      2.6954637  0.2506277  2.2446483
##   0.68421053  0.30      2.6434476  0.2840811  2.1990228
##   0.68421053  0.35      2.5935834  0.3154645  2.1552952
##   0.68421053  0.40      2.5458999  0.3443489  2.1131843
##   0.68421053  0.45      2.5004421  0.3704617  2.0720233
##   0.68421053  0.50      2.4576757  0.3935646  2.0320836
##   0.68421053  0.55      2.4174131  0.4139440  1.9935737
##   0.68421053  0.60      2.3798574  0.4316519  1.9565389
##   0.68421053  0.65      2.3449113  0.4470421  1.9219717
##   0.68421053  0.70      2.3125751  0.4603000  1.8901265
##   0.68421053  0.75      2.2832404  0.4715635  1.8601649
##   0.68421053  0.80      2.2569393  0.4810494  1.8322606
##   0.68421053  0.85      2.2341767  0.4887792  1.8075558
##   0.68421053  0.90      2.2145036  0.4951365  1.7854592
##   0.68421053  0.95      2.1977047  0.5002994  1.7657522
##   0.68421053  1.00      2.1838929  0.5042655  1.7486696
##   0.73684211  0.05      2.8927037  0.1154780  2.4409378
##   0.73684211  0.10      2.8712000  0.1420258  2.3956243
##   0.73684211  0.15      2.8138658  0.1751108  2.3447536
##   0.73684211  0.20      2.7583690  0.2089513  2.2964308
##   0.73684211  0.25      2.7050803  0.2423304  2.2499022
##   0.73684211  0.30      2.6539772  0.2744490  2.2050664
##   0.73684211  0.35      2.6050136  0.3047716  2.1618581
##   0.73684211  0.40      2.5581627  0.3329173  2.1205962
##   0.73684211  0.45      2.5135607  0.3585360  2.0802169
##   0.73684211  0.50      2.4716125  0.3813811  2.0411816
##   0.73684211  0.55      2.4321027  0.4016903  2.0035430
##   0.73684211  0.60      2.3952693  0.4194550  1.9673785
##   0.73684211  0.65      2.3609900  0.4349891  1.9338810
##   0.73684211  0.70      2.3293014  0.4484348  1.9026435
##   0.73684211  0.75      2.3005531  0.4599130  1.8734153
##   0.73684211  0.80      2.2747451  0.4696404  1.8460568
##   0.73684211  0.85      2.2524149  0.4776065  1.8216734
##   0.73684211  0.90      2.2330515  0.4842047  1.7998752
##   0.73684211  0.95      2.2164642  0.4896163  1.7805177
##   0.73684211  1.00      2.2026732  0.4938707  1.7638205
##   0.78947368  0.05      2.8923879  0.1154044  2.4423960
##   0.78947368  0.10      2.8778497  0.1392223  2.3974376
##   0.78947368  0.15      2.8214902  0.1706749  2.3476298
##   0.78947368  0.20      2.7669032  0.2029207  2.3004415
##   0.78947368  0.25      2.7145109  0.2348203  2.2549062
##   0.78947368  0.30      2.6642438  0.2656831  2.2108044
##   0.78947368  0.35      2.6161032  0.2949852  2.1682881
##   0.78947368  0.40      2.5700310  0.3223741  2.1274762
##   0.78947368  0.45      2.5262083  0.3474700  2.0878104
##   0.78947368  0.50      2.4850055  0.3700133  2.0496863
##   0.78947368  0.55      2.4461917  0.3901935  2.0127999
##   0.78947368  0.60      2.4100351  0.4079522  1.9774685
##   0.78947368  0.65      2.3763796  0.4235761  1.9450060
##   0.78947368  0.70      2.3452978  0.4371619  1.9144500
##   0.78947368  0.75      2.3171089  0.4488147  1.8858580
##   0.78947368  0.80      2.2917650  0.4587551  1.8591582
##   0.78947368  0.85      2.2698360  0.4669387  1.8350555
##   0.78947368  0.90      2.2507757  0.4737572  1.8133779
##   0.78947368  0.95      2.2343908  0.4794047  1.7942223
##   0.78947368  1.00      2.2206353  0.4839314  1.7780568
##   0.84210526  0.05      2.8921287  0.1153372  2.4436931
##   0.84210526  0.10      2.8845985  0.1366815  2.3992632
##   0.84210526  0.15      2.8291450  0.1666632  2.3504173
##   0.84210526  0.20      2.7754052  0.1974493  2.3043138
##   0.84210526  0.25      2.7238392  0.2279871  2.2596868
##   0.84210526  0.30      2.6743348  0.2576763  2.2163731
##   0.84210526  0.35      2.6269533  0.2859954  2.1746506
##   0.84210526  0.40      2.5816084  0.3126265  2.1340657
##   0.84210526  0.45      2.5385165  0.3371717  2.0949803
##   0.84210526  0.50      2.4979933  0.3593801  2.0576871
##   0.84210526  0.55      2.4598352  0.3793786  2.0215378
##   0.84210526  0.60      2.4243131  0.3970810  1.9871644
##   0.84210526  0.65      2.3912445  0.4127468  1.9555317
##   0.84210526  0.70      2.3607328  0.4264319  1.9256916
##   0.84210526  0.75      2.3330688  0.4382260  1.8976040
##   0.84210526  0.80      2.3081655  0.4483503  1.8714906
##   0.84210526  0.85      2.2866174  0.4567275  1.8478082
##   0.84210526  0.90      2.2678466  0.4637506  1.8262652
##   0.84210526  0.95      2.2516604  0.4696178  1.8074152
##   0.84210526  1.00      2.2379526  0.4744009  1.7918705
##   0.89473684  0.05      2.8919143  0.1152753  2.4448530
##   0.89473684  0.10      2.8914693  0.1343653  2.4013062
##   0.89473684  0.15      2.8368644  0.1630143  2.3534126
##   0.89473684  0.20      2.7839169  0.1924621  2.3081110
##   0.89473684  0.25      2.7331196  0.2217422  2.2643021
##   0.89473684  0.30      2.6843312  0.2503269  2.2217564
##   0.89473684  0.35      2.6376434  0.2777131  2.1806877
##   0.89473684  0.40      2.5929753  0.3035976  2.1406430
##   0.89473684  0.45      2.5505807  0.3275689  2.1021223
##   0.89473684  0.50      2.5107070  0.3494002  2.0653962
##   0.89473684  0.55      2.4731567  0.3691823  2.0299206
##   0.89473684  0.60      2.4382274  0.3867894  1.9964946
##   0.89473684  0.65      2.4057130  0.4024544  1.9655374
##   0.89473684  0.70      2.3757375  0.4162020  1.9363694
##   0.89473684  0.75      2.3485703  0.4281054  1.9088985
##   0.89473684  0.80      2.3240928  0.4383843  1.8832313
##   0.89473684  0.85      2.3029035  0.4469334  1.8600481
##   0.89473684  0.90      2.2844055  0.4541467  1.8389752
##   0.89473684  0.95      2.2684133  0.4602179  1.8207275
##   0.89473684  1.00      2.2547651  0.4652412  1.8054722
##   0.94736842  0.05      2.8917337  0.1152178  2.4458929
##   0.94736842  0.10      2.8984743  0.1322493  2.4034654
##   0.94736842  0.15      2.8446725  0.1596811  2.3564798
##   0.94736842  0.20      2.7924645  0.1879028  2.3118129
##   0.94736842  0.25      2.7423901  0.2160164  2.2688313
##   0.94736842  0.30      2.6942816  0.2435595  2.2270706
##   0.94736842  0.35      2.6482425  0.2700553  2.1866143
##   0.94736842  0.40      2.6042065  0.2952121  2.1471777
##   0.94736842  0.45      2.5624726  0.3186013  2.1093690
##   0.94736842  0.50      2.5231986  0.3400419  2.0730297
##   0.94736842  0.55      2.4862308  0.3595675  2.0380020
##   0.94736842  0.60      2.4518623  0.3770406  2.0054789
##   0.94736842  0.65      2.4198722  0.3926681  1.9752032
##   0.94736842  0.70      2.3904160  0.4064379  1.9466404
##   0.94736842  0.75      2.3637187  0.4184224  1.9197493
##   0.94736842  0.80      2.3396502  0.4288278  1.8946745
##   0.94736842  0.85      2.3187999  0.4375294  1.8721271
##   0.94736842  0.90      2.3005668  0.4449146  1.8516479
##   0.94736842  0.95      2.2847618  0.4511752  1.8338874
##   0.94736842  1.00      2.2711866  0.4564213  1.8189786
##   1.00000000  0.05      2.8915840  0.1151640  2.4468335
##   1.00000000  0.10      2.9056214  0.1303071  2.4058371
##   1.00000000  0.15      2.8525888  0.1566221  2.3596452
##   1.00000000  0.20      2.8010840  0.1837122  2.3156463
##   1.00000000  0.25      2.7516870  0.2107468  2.2733637
##   1.00000000  0.30      2.7042238  0.2373101  2.2324739
##   1.00000000  0.35      2.6588068  0.2629514  2.1924381
##   1.00000000  0.40      2.6153672  0.2874000  2.1537403
##   1.00000000  0.45      2.5742467  0.3102192  2.1165965
##   1.00000000  0.50      2.5355461  0.3312471  2.0807282
##   1.00000000  0.55      2.4991385  0.3504881  2.0460318
##   1.00000000  0.60      2.4652996  0.3677969  2.0143317
##   1.00000000  0.65      2.4338222  0.3833455  1.9846325
##   1.00000000  0.70      2.4048557  0.3971108  1.9566516
##   1.00000000  0.75      2.3786084  0.4091471  1.9306799
##   1.00000000  0.80      2.3549307  0.4196553  1.9063240
##   1.00000000  0.85      2.3344096  0.4284858  1.8840705
##   1.00000000  0.90      2.3164255  0.4360287  1.8641140
##   1.00000000  0.95      2.3008062  0.4424624  1.8470821
##   1.00000000  1.00      2.2873105  0.4479154  1.8323720
##
## 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.
```

```
resamp = resamples(list(lm=lm.model,rlm=rlm.model,ridge=ridge.model,enet=enet.model) )
print( summary(resamp) )
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: lm, rlm, ridge, enet
## Number of resamples: 50
##
## MAE
##            Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## lm    0.4249245 0.6008457 0.7025946 0.7567567 0.8537570 1.257947    0
## rlm   1.5693402 1.9915552 2.1085728 2.1465819 2.3325043 2.890485    0
## ridge 0.3257362 0.5571046 0.6880492 0.7342679 0.8794620 1.367363    0
## enet  0.2812595 0.4261824 0.5035478 0.5308600 0.6053735 1.011333    0
##
## RMSE
##            Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## lm    0.5645011 0.8159497 1.0212984 1.2141113 1.3954875 3.007845    0
## rlm   2.0715113 2.4230593 2.6417904 2.6394450 2.8061668 3.356444    0
## ridge 0.4224940 0.7540424 1.0101131 1.1938739 1.4258913 3.299728    0
## enet  0.3381878 0.5536817 0.6542294 0.7762276 0.7980329 1.826297    0
##
## Rsquared
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lm    0.183083911 0.8562545 0.9033451 0.8403278 0.9290080 0.9673908    0
## rlm   0.003241385 0.1713507 0.2435282 0.2590020 0.3500647 0.5343185    0
## ridge 0.214240742 0.8267331 0.9098722 0.8476045 0.9426539 0.9906607    0
## enet  0.668189649 0.9442637 0.9531603 0.9298649 0.9713312 0.9898207    0
```

```
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
##       lm        rlm       ridge     enet
## lm              -1.38983   0.02249   0.22590
## rlm   < 2.2e-16            1.41231   1.61572
## ridge 1         < 2.2e-16            0.20341
## enet  3.866e-06 < 2.2e-16 9.274e-08
##
## RMSE
##       lm        rlm       ridge     enet
## lm              -1.42533   0.02024   0.43788
## rlm   < 2.2e-16            1.44557   1.86322
## ridge 1.0000000 < 2.2e-16            0.41765
## enet  0.0007065 < 2.2e-16 4.175e-06
##
## Rsquared
##       lm        rlm       ridge     enet
## lm               0.581326 -0.007277 -0.089537
## rlm   < 2.2e-16           -0.588603 -0.670863
## ridge 1.000000  < 2.2e-16           -0.082260
## enet  0.006676  < 2.2e-16 0.001632
```

So, the different models that I have tried are **Simple Linear Regression**, **Robust Linear Regression**, **Ridge Regression** and **Elastic Net**. The models **Ridge Regression** and **Elastic Net* have tuning parameters. For** Ridge Regression**, the optimal value for the tuning parameter (λ) is 0, RMSE is the lowest 1.193874, $\text{R} - \text{squared}$ is the highest 0.8476045.

For **Elastic Net*, the optimal values for $\text{fraction} = 0.05$ and $\lambda = 0$, RMSE is the lowest 0.7762276, $\text{R} - \text{squared}$ is the highest 0.9298649.

6.1 d

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

Answer (6.1 d)

```
# Prediction for Simple Linear Model and MSE

MSE.lm.test <- mean((predict(lm.model, data=absorp.test) - protein.test)^2)
MSE.lm.test
```

```
## [1] 19.53647
```

```
RMSE.lm.test = sqrt(MSE.lm.test)
RMSE.lm.test
```

```
## [1] 4.420008
```

```
# Prediction for Robust Linear Regression Model and MSE

MSE.rlm.test <- mean((predict(rlm.model, data=absorp.test) - protein.test)^2)
MSE.rlm.test
```

```
## [1] 11.4929
```

```
RMSE.rlm.test = sqrt(MSE.rlm.test)
RMSE.rlm.test
```

```
## [1] 3.390118
```

```
# Prediction for Ridge Regression Model and RMSE

MSE.ridge.test <- mean((predict(ridge.model, data=absorp.test) - protein.test)^2)
MSE.ridge.test
```

```
## [1] 19.53647
```

```
RMSE.ridge.test = sqrt(MSE.ridge.test)
RMSE.ridge.test
```

```
## [1] 4.420008
```

```
# Prediction for Elastic Net Model and RMSE

MSE.enet.test <- mean((predict(enet.model, data=absorp.test) - protein.test)^2)
MSE.enet.test
```

```
## [1] 19.43613
```

```
RMSE.enet.test = sqrt(MSE.enet.test)
RMSE.enet.test
```

```
## [1] 4.408643
```

Based on the results of the predictions, **Robust Linear Regression Model** has the lowest MSE of 11.4929 and so this is the model that is a best fit.

6.1 e

Explain which model you would use for predicting the percentage of protein of a sample.

Answer (6.1 e)

Based on the results obtained, I would use the **Robust Linear Regression Model** for predicting the protein of a sample as we got the lowest MSE for that model.

Q 6.2

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:

6.2 a

Start R and use these commands to load the data

Answer (6.2 a)

```
library(AppliedPredictiveModeling)
data(permeability)
```

The matrix **fingerprints** contains the 1,107 binary molecular predictors for the 165 compounds, while **permeability** contains permeability response.

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?

Answer (6.2 b)

```
# Number of predictors in fingerprints before non-zero variance

print(str(fingerprints))
```

```
##  num [1:165, 1:1107] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:165] "1" "2" "3" "4" ...
##   ..$ : chr [1:1107] "X1" "X2" "X3" "X4" ...
## NULL
```

```
cat("Before Non-Zero Variance, number of predictors in fingerprints is 1107: \n")
```

```
## Before Non-Zero Variance, number of predictors in fingerprints is 1107:
```

```
# Number of predictors in fingerprints after non-zero variance

NZVfingerprints <- nearZeroVar(fingerprints)
noNZVfingerprints <- fingerprints[,-NZVfingerprints]
print(str(NZVfingerprints))
```

```
##  int [1:719] 7 8 9 10 13 14 17 18 19 22 ...
## NULL
```

```
print(str(noNZVfingerprints))
```

```
##  num [1:165, 1:388] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:165] "1" "2" "3" "4" ...
##   ..$ : chr [1:388] "X1" "X2" "X3" "X4" ...
## NULL
```

```
cat("After Non-Zero Variance, number of predictors in fingerprints is 388: \n")
```

```
## After Non-Zero Variance, number of predictors in fingerprints is 388:
```

So, we find from above that there are in total 1107 predictors. Out of these, there are **719** near-zero variance fingerprints, which means that there are **388** predictors that are left for modeling. This means that around **35%** of the total predictors are left for modelling and this is a significant reduction from the original matrix. This is also an indication of the fact that many fingerprints are describing unique features of very small subsets.

Now, because this data set has small number of samples in comparison to predictors $(n = 165, p = 388)$, we think that a **75% training** and **25% test** split is ideal to evaluate model performance.

6.2 c

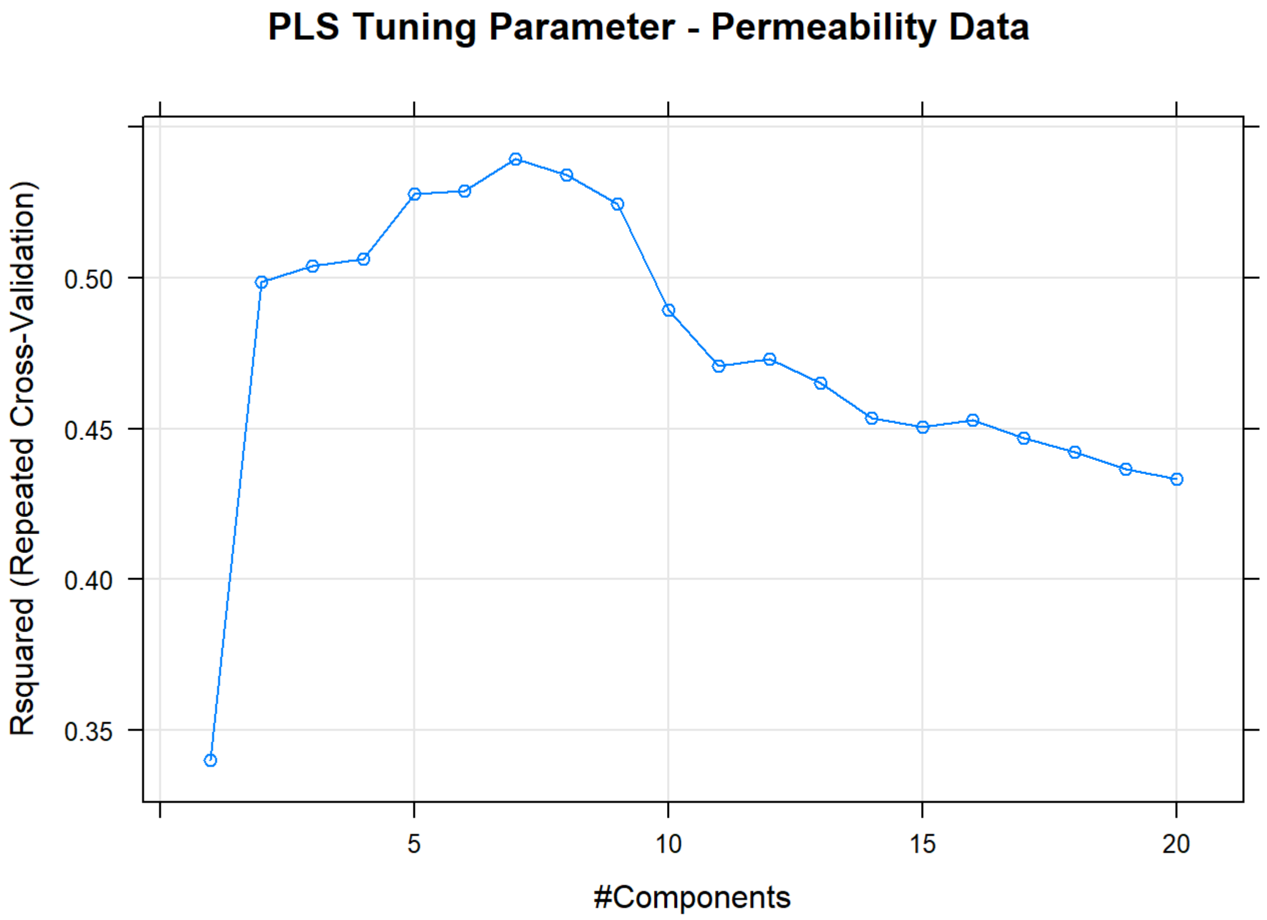
Split the data into a training and a test set, pre-process the data, and tune a PLS model. How many latent variables are optimal and what is the corresponding re sampled estimate of R^2?

Answer (6.2 c)

```
# Splitting data into 75% training and 25% test dataset


set.seed(123)
trainRows =  createDataPartition(permeability, p = .75, list= FALSE)

trainFingerprints <- noNZVfingerprints[trainRows,]
trainPermeability <- permeability[trainRows,]

testFingerprints <- noNZVfingerprints[-trainRows,]
testPermeability <- permeability[-trainRows,]

set.seed(123)

ctrl <- trainControl(method = "repeatedcv", repeats=5, number = 10)




# PLS Model

pls.model <- train(x = trainFingerprints , y = trainPermeability,preProcess = c("center","scale"), method = "pls", tuneGrid
 = expand.grid(ncomp = 1:20), trControl = ctrl)
print(pls.model)
```

```
## Partial Least Squares
##
## 125 samples
## 388 predictors
##
## Pre-processing: centered (388), scaled (388)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 111, 112, 113, 113, 113, 113, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    1     13.36189  0.3400650  10.235098
##    2     11.69727  0.4987534   8.409279
##    3     11.70107  0.5040951   8.872360
##    4     11.73732  0.5064997   8.958828
##    5     11.50726  0.5277856   8.550770
##    6     11.52201  0.5289935   8.460554
##    7     11.49204  0.5395630   8.665686
##    8     11.62383  0.5343839   8.969844
##    9     11.88929  0.5245591   9.211200
##   10     12.45459  0.4894157   9.634534
##   11     12.78770  0.4709986   9.751598
##   12     12.83110  0.4732919   9.748033
##   13     12.92887  0.4651002   9.777449
##   14     13.03178  0.4534699   9.795611
##   15     13.07274  0.4506160   9.774092
##   16     13.04306  0.4528448   9.823553
##   17     13.17445  0.4470634   9.870590
##   18     13.28197  0.4421497   9.928353
##   19     13.34085  0.4367930  10.002830
##   20     13.43762  0.4334184  10.076034
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 7.
```

```
plot(pls.model, metric ="Rsquared", main = "PLS Tuning Parameter - Permeability Data")
```



Based on the outcome of the PLS model, we find out that the optimal value of ncomp commonly known as latent variables is 2. The corresponding R^2 is the highest and is 0.4578040

6.2 d

Predict the response for the test set. What is the test set estimate of R^2?

Answer (6.2 d)

```
# Prediction performance using PLS based on the test set

pred.pls = predict( pls.model, newdata=testFingerprints )
rsq.pls = cor(pred.pls,testPermeability,method="pearson")^2
rmse.pls = sqrt( mean( (pred.pls - testPermeability)^2 ) )
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "PLS", rsq.pls, rmse.pls ) )
```

```
## [1] "PLS       : Testing R-square =   0.404734; RMSE=  11.592026"
```

6.2 e

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

Answer (6.2 e)

```
# Elastic Net



enetGrid = expand.grid(.lambda=seq(0,1,length=20), .fraction=seq(0.05, 1.0, length=20))
set.seed(0)
enet.model = train( trainFingerprints, trainPermeability, method="enet",
                    # fit the model over many penalty values
                    tuneGrid = enetGrid,
                    preProcess=c("center","scale"), trControl=trainControl(method="repeatedcv",repeats=5) )

pred.enet = predict( enet.model, newdata=testFingerprints )
rsq.enet = cor(pred.enet,testPermeability,method="pearson")^2
rmse.enet = sqrt( mean( (pred.enet - testPermeability)^2 ) )
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "ENET", rsq.enet, rmse.enet ) )
```

```
## [1] "ENET      : Testing R-square =   0.317369; RMSE=  12.039292"
```

```
# Linear Regression Model

set.seed(0)
lm.model = train( trainFingerprints, trainPermeability, method="lm", preProcess=c("center","scale"), trControl=trainControl
(method="repeatedcv",repeats=5) )

pred.lm = predict( lm.model, newdata=testFingerprints )
rsq.lm = cor(pred.lm,testPermeability,method="pearson")^2
rmse.lm = sqrt( mean( (pred.lm - testPermeability)^2 ) )
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "LM", rsq.lm, rmse.lm ) )
```

```
## [1] "LM        : Testing R-square =   0.008443; RMSE=  38.085941"
```

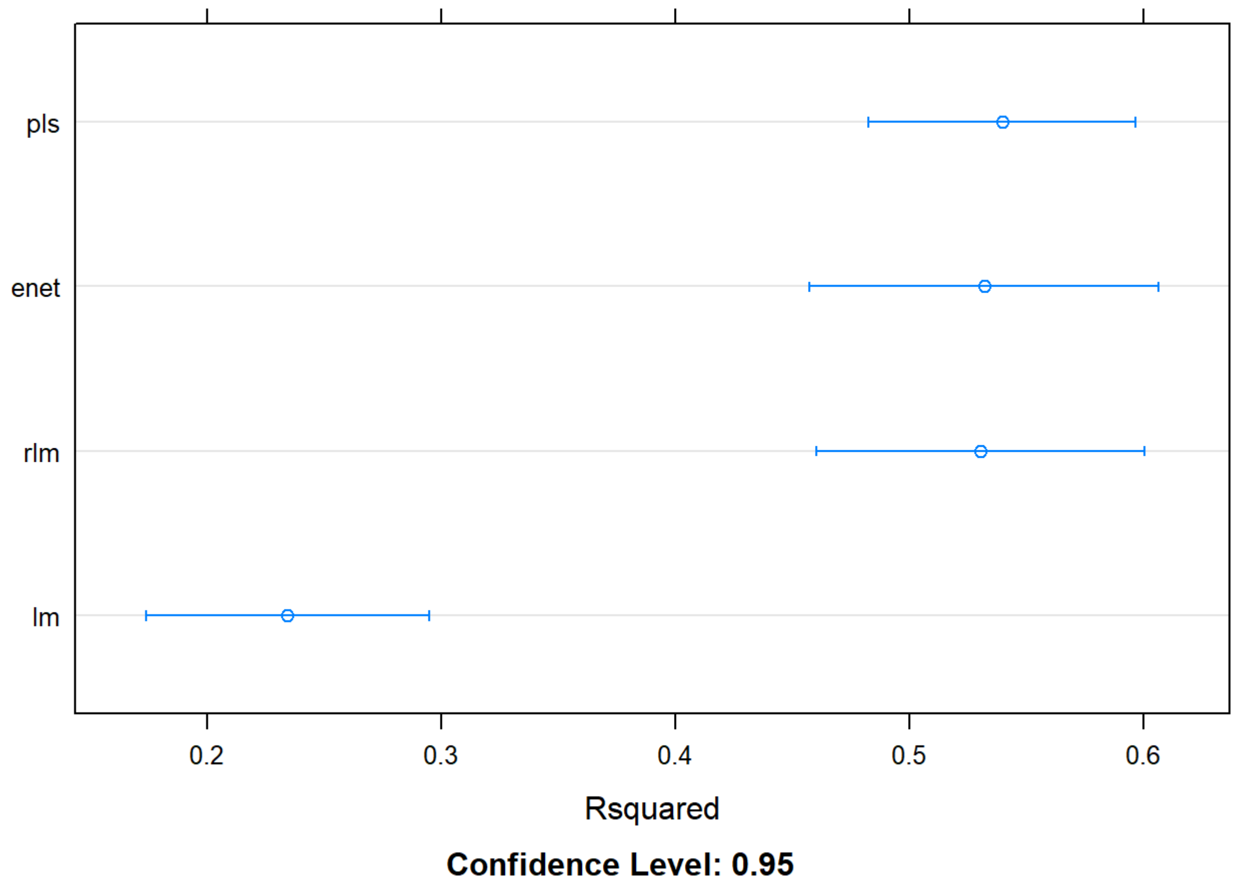
```
# Robust Linear Regression. For RLM, we cannot have a singular predictor covariance matrix and so we preprocess with PCA:
#
set.seed(0)
rlm.model = train( trainFingerprints, trainPermeability, method="rlm", preProcess=c("pca"), trControl=trainControl(method="r
epeatedcv",repeats=5) )

pred.rlm = predict( rlm.model, newdata=testFingerprints )
rsq.rlm = cor(pred.rlm,testPermeability,method="pearson")^2
rmse.rlm = sqrt( mean( (pred.rlm - testPermeability)^2 ) )
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "RLM", rsq.rlm, rmse.rlm ) )
```

```
## [1] "RLM       : Testing R-square =   0.369715; RMSE=  12.055161"
```

```
# Comparison of the above models using re samples

resamp = resamples( list(pls=pls.model,lm=lm.model, rlm=rlm.model, enet = enet.model) )
print( summary(resamp) )
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: pls, lm, rlm, enet
## Number of resamples: 50
##
## MAE
##          Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## pls  5.440095  7.301233  8.379453  8.665686  9.773692 15.01704    0
## lm   7.914729 13.179356 16.865817 18.520286 21.578824 50.68060    0
## rlm  4.213104  6.643318  7.549829  7.866678  8.841155 16.24117    0
## enet 4.033392  6.190007  7.720733  7.603388  8.721203 14.22762    0
##
## RMSE
##          Min.   1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## pls  7.336246  9.602985 11.60060 11.49204 12.41528 18.98603    0
## lm   9.934366 17.993800 25.80763 27.02225 33.84548 81.51337    0
## rlm  5.300764  9.118978 10.94546 11.36327 13.18573 22.05604    0
## enet 5.253304  8.647688 12.06243 11.17971 13.00288 19.94595    0
##
## Rsquared
##              Min.    1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## pls  0.0336063297 0.45070969 0.5507040 0.5395630 0.6625829 0.9040862    0
## lm   0.0006005043 0.04787819 0.1885445 0.2341566 0.3934569 0.7725173    0
## rlm  0.0061292592 0.36750419 0.5801461 0.5301563 0.7354222 0.9117009    0
## enet 0.0140910435 0.32510952 0.5557068 0.5318608 0.7589479 0.9569953    0
```

```
dotplot( resamp, metric="Rsquared" )
```



```
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       lm        rlm     enet
## pls            -9.8546    0.7990  1.0623
## lm   6.717e-11           10.6536 10.9169
## rlm  0.2806    1.366e-11          0.2633
## enet 0.0480    5.538e-12 1.0000
##
## RMSE
##      pls       lm         rlm      enet
## pls            -15.5302    0.1288   0.3123
## lm   4.805e-11            15.6590  15.8425
## rlm  1         5.428e-11            0.1836
## enet 1         1.272e-11 1
##
## Rsquared
##      pls       lm        rlm       enet
## pls             0.305406  0.009407  0.007702
## lm   1.202e-08           -0.296000 -0.297704
## rlm  1         1.200e-10           -0.001704
## enet 1         8.592e-11 1
```

6.2 f

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

Answer (6.2 f)

Yes, I would recommend using **Elastic Net** model as it gives me the **highest** R^2

Chapter 7 - E-Book - Applied Predictive Modelling - Exercises pages 170-171:

Q 7.4

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

```
set.seed(0)
trainRows =  createDataPartition(permeability, p = .75)

trainFingerprints <- noNZVfingerprints[trainRows$Resample1,]
trainPermeability <- permeability[trainRows$Resample1,]

testFingerprints <- noNZVfingerprints[-trainRows$Resample1,]
testPermeability <- permeability[-trainRows$Resample1,]
```

```
# A K-NN model



set.seed(0)
knn.model = train(x=trainFingerprints, y=trainPermeability, method="knn", preProcess=c("center","scale"), tuneLength=10)

# Prediction on testing dataset

pred.knn = predict(knn.model, newdata=testFingerprints)
pr.knn = postResample(pred=pred.knn, obs=testPermeability)
rmse.knn = c(pr.knn[1])
rsq.knn = c(pr.knn[2])
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "KNN", rsq.knn, rmse.knn ) )
```

```
## [1] "KNN       : Testing R-square =   0.690080; RMSE=   8.924366"
```

```
# Neural Network model:

nnGrid = expand.grid( .decay=c(0,0.01,0.1), .size=1:10, .bag=FALSE )
set.seed(0)
nnet.model = train(x=trainFingerprints, y=trainPermeability, method="nnet", preProcess=c("center","scale"),
                  linout=TRUE,trace=FALSE,MaxNWts=10 * (ncol(trainFingerprints)+1) + 10 + 1, maxit=500)


# Prediction on testing dataset

pred.nnet = predict(nnet.model, newdata=testFingerprints)
pr.nnet = postResample(pred=pred.nnet, obs=testPermeability)
rmse.nnet = c(pr.nnet[1])
rsq.nnet = c(pr.nnet[2])
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "Neural Net", rsq.nnet, rmse.nnet ) )
```

```
## [1] "Neural Net: Testing R-square =   0.377918; RMSE=  13.156465"
```

```
# MARS model:

marsGrid = expand.grid(.degree=1:2, .nprune=2:38)
set.seed(0)
mars.model = train(x=trainFingerprints, y=trainPermeability, method="earth", preProcess=c("center","scale"), tuneGrid=marsGr
id)


# Prediction on testing dataset

pred.mars = predict(mars.model, newdata=testFingerprints)
pr.mars = postResample(pred=pred.mars, obs=testPermeability)
rmse.mars = c(pr.mars[1])
rsq.mars = c(pr.mars[2])
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "MARS", rsq.mars, rmse.mars ) )
```

```
## [1] "MARS      : Testing R-square =   0.578509; RMSE=  10.722704"
```

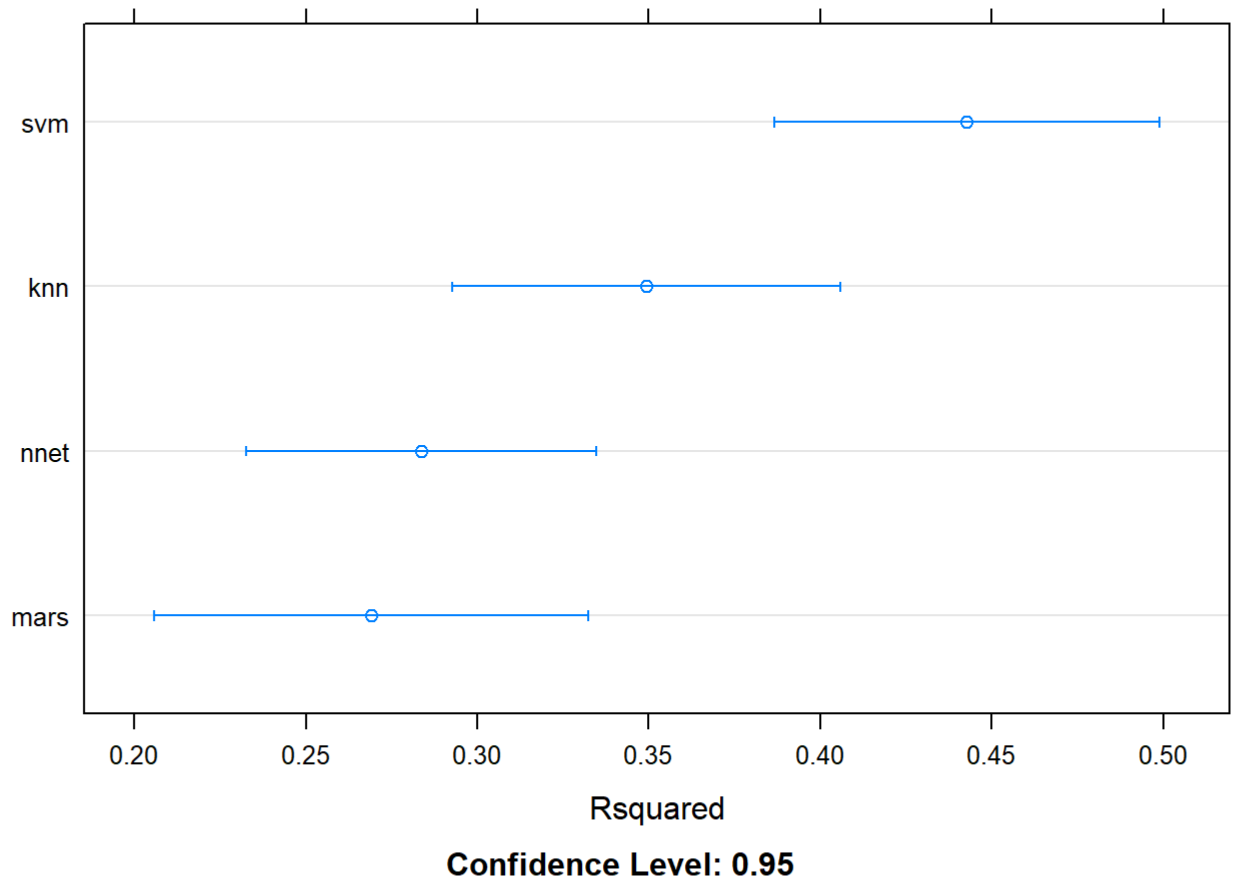
```
# A Support Vector Machine (SVM):

set.seed(0)
svm.model = train(x=trainFingerprints, y=trainPermeability, method="svmRadial", preProcess=c("center","scale"), tuneLength=2
0)



# Prediction on testing dataset

pred.svm = predict(svm.model, newdata=testFingerprints)
pr.svm = postResample(pred=pred.svm, obs=testPermeability)
rmse.svm = c(pr.svm[1])
rsq.svm = c(pr.svm[2])
print( sprintf( "%-10s: Testing R-square = %10.6f; RMSE= %10.6f", "SVM Radial", rsq.svm, rmse.svm ) )
```

```
## [1] "SVM Radial: Testing R-square =   0.668277; RMSE=   9.236457"
```

```
# Comparison of the above models using re samples

resamp = resamples( list(svm=svm.model,knn=knn.model,nnet=nnet.model,mars=mars.model) )
print( summary(resamp) )
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: svm, knn, nnet, mars
## Number of resamples: 25
##
## MAE
##          Min.  1st Qu.   Median      Mean   3rd Qu.     Max. NA's
## svm  6.340643 7.683391 8.203117  8.321979  8.704631 10.50581    0
## knn  6.854129 7.698445 9.070021  8.823619  9.731999 11.21189    0
## nnet 7.592364 9.402606 9.930507 10.299609 11.092731 14.68836    0
## mars 7.389179 9.242985 9.844095  9.953571 10.671004 13.38020    0
##
## RMSE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## svm   8.631494 10.63593 11.94560 11.80441 12.97691 15.13842    0
## knn  10.375440 11.24613 13.47294 12.92734 14.06650 17.03005    0
## nnet 10.245726 12.62929 13.90086 13.99409 14.92288 19.16586    0
## mars  9.954479 12.93607 13.52691 13.65413 14.12911 18.80341    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## svm  0.183001417 0.3412150 0.4318521 0.4427593 0.5704215 0.6314768    0
## knn  0.074902934 0.2558160 0.3157898 0.3492673 0.4771551 0.5901835    0
## nnet 0.062123235 0.1975906 0.2683109 0.2837265 0.3861917 0.5619591    0
## mars 0.002790901 0.1827847 0.2952871 0.2690972 0.3583866 0.6260089    0
```

```
dotplot( resamp, metric="Rsquared" )
```



```
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
##      svm       knn       nnet     mars
## svm            -0.5016   -1.9776  -1.6316
## knn  0.050918            -1.4760  -1.1300
## nnet 5.414e-08 6.089e-06           0.3460
## mars 4.304e-06 0.001028  1.000000
##
## RMSE
##      svm       knn      nnet     mars
## svm            -1.1229  -2.1897  -1.8497
## knn  8.143e-06          -1.0668  -0.7268
## nnet 6.986e-07 0.004726           0.3400
## mars 7.886e-05 0.236104 1.000000
##
## Rsquared
##      svm       knn     nnet    mars
## svm            0.09349 0.15903 0.17366
## knn  5.072e-06         0.06554 0.08017
## nnet 6.067e-06 0.1229          0.01463
## mars 8.414e-05 0.1489  1.0000
```

7.4 a

Which nonlinear regression model gives the optimal re sampling and test set performance?

Answer (7.4 a)

Based on the outcomes above, **Support Vector Machine (SVM)** provides the optimal re sampling and test performance.

7.4 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?

Answer (7.4 b)

Yes, **Support Vector Machine (SVM)** does outperform the other linear models , because of all these models the RMSE is **least**.

7.4 c

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

Answer (7.4 c)

I would recommend using **Support Vector Machine (SVM)** model.