Problem Set 4: Biased Methods and Regularization

Donggyun Kim
27008257

3/21/2018

1) Properties of PLS Regression

a)

$||\mathbf{w}_h|| = \mathbf{w}_h^t \mathbf{w}_h = 1.$

$\mathbf{p}_h = \frac{\mathbf{X}_{h-1}^t \mathbf{z}_h}{\mathbf{z}_h^t \mathbf{z}_h} = \frac{\mathbf{X}_{h-1}^t \mathbf{X}_{h-1} \mathbf{w}_h}{\mathbf{w}_h^t \mathbf{X}_{h-1}^t \mathbf{X}_{h-1} \mathbf{w}_h}.$

$\mathbf{w}_h^t \mathbf{p}_h = \frac{\mathbf{w}_h^t \mathbf{X}_{h-1}^t \mathbf{X}_{h-1} \mathbf{w}_h}{\mathbf{w}_h^t \mathbf{X}_{h-1}^t \mathbf{X}_{h-1} \mathbf{w}_h} = 1.$

b)

$\mathbf{X}_h = \mathbf{X}_{h-1} - \mathbf{z}_h \mathbf{p}_h^t = \mathbf{X}_{h-1} - \mathbf{X}_{h-1} \mathbf{w}_h \mathbf{p}_h^t.$

$\mathbf{X}_h \mathbf{w}_h = \mathbf{X}_{h-1} \mathbf{w}_h - \mathbf{X}_{h-1} \mathbf{w}_h \mathbf{p}_h^t \mathbf{w}_h.$

$\mathbf{w}_h^t \mathbf{X}_h^t = (\mathbf{X}_h \mathbf{w}_h)^t = \mathbf{w}_h^t \mathbf{X}_{h-1}^t - \mathbf{w}_h^t \mathbf{p}_h \mathbf{w}_h^t \mathbf{X}_{h-1}^t = \mathbf{w}_h^t \mathbf{X}_{h-1}^t - \mathbf{w}_h^t \mathbf{X}_{h-1}^t = 0.$

2) Bias of Regression Coefficients in PCR

a)

$\hat{\beta}_Z^{(k)} = V_k \hat{\beta}_{PCR}^{(k)}.$

$\mathbb{E}[V_k \hat{\beta}_{PCR}^{(k)}] = V_k (Z_k^t Z_k)^{-1} Z_k^t \mathbb{E}[y] = V_k (\Lambda_k)^{-1} (X V_k)^t (X\beta) = V_k (\Lambda_k)^{-1} V_k^t X^t X \beta.$

$\mathbb{E}[V_k \hat{\beta}_{PCR}^{(k)} - \beta] = \mathbb{E}[V_k \hat{\beta}_{PCR}^{(k)}] - \beta = V_k (\Lambda_k)^{-1} V_k^t X^t X \beta - \beta = (V_k (\Lambda_k)^{-1} V_k^t X^t X - I)\beta.$

b)

$\hat{\beta}_Z^{(p)} = V_p \hat{\beta}_{PCR}^{(p)}.$

$\mathbb{E}[V_p \hat{\beta}_{PCR}^{(p)}] = V_p (Z_p^t Z_p)^{-1} Z_p^t \mathbb{E}[y] = V_p ((X V_p)^t X V_p)^{-1} (X V_p)^t (X\beta) = V_p (V_p^t X^t X V_p)^{-1} V_p^t X^t X \beta = V_p V_p^t (X^t X)^{-1} V_p V_p^t X^t X \beta = \beta.$

$\mathbb{E}[V_p \hat{\beta}_{PCR}^{(p)} - \beta] = 0.$

3) Bias of Ridge Regression Coefficients

$X = UDV^t$.

$\hat{\beta}_r = (X^tX + kI)^{-1}X^ty = ((UDV^t)^tUDV^t + kI)^{-1}(UDV^t)^ty = (VD^2V^t + kI)^{-1}VDU^ty$.

$\mathbb{E}[\hat{\beta}_r] = (VD^2V^t + kI)^{-1}VDU^tX\beta = (VD^2V^t + kI)^{-1}VDU^tUDV^t\beta = (VD^2V^t + kI)^{-1}VD^2V^t\beta$.

$D^2 = \Lambda$.

$\mathbb{E}[\hat{\beta}_r - \beta] = \mathbb{E}[\hat{\beta}_r] - \beta = (V\Lambda V^t + kI)^{-1}V\Lambda V^t\beta - \beta = ((V\Lambda V^t + kI)^{-1}V\Lambda V^t - I)\beta$.

4) Models for Solubility Data

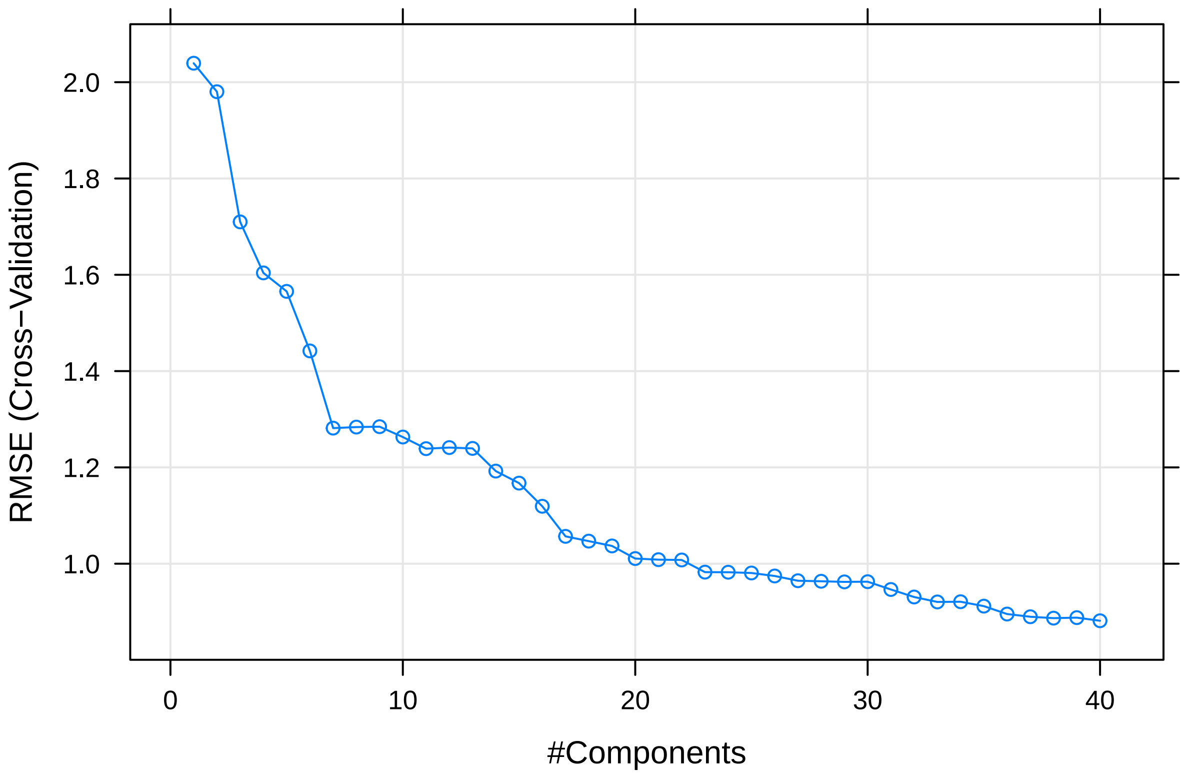
4.1) PCR

```r
library(AppliedPredictiveModeling)
library(caret)
library(pls)
library(elasticnet)
library(ggplot2)

data(solubility)

# 10-fold cross-validation
ctrl <- trainControl(method = "cv", number = 10)

set.seed(1991)
pcr_fit <- train(x = solTrainXtrans, y = solTrainY,
                 method = "pcr",
                 tuneLength = 40,
                 trControl = ctrl,
                 preProcess = c("center", "scale"))
pcr_fit
```

```
## Principal Component Analysis
##
## 951 samples
## 228 predictors
##
## Pre-processing: centered (228), scaled (228)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 855, 857, 855, 857, 857, 855, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared    MAE
##    1     2.0393844  0.01330618  1.5768251
##    2     1.9802436  0.08618857  1.5600094
##    3     1.7098785  0.30775962  1.3508311
##    4     1.6039381  0.38902366  1.2491550
##    5     1.5655620  0.41438454  1.2160916
##    6     1.4419644  0.50548830  1.1133483
##    7     1.2816012  0.60844627  1.0008166
```

```
##    8      1.2837318  0.60644396  1.0030821
##    9      1.2845568  0.60575993  1.0043077
##   10      1.2630788  0.61833048  0.9805825
##   11      1.2389026  0.63310862  0.9595090
##   12      1.2411704  0.63155212  0.9651889
##   13      1.2394323  0.63270101  0.9608776
##   14      1.1923383  0.66024430  0.9291284
##   15      1.1673741  0.67410421  0.9143805
##   16      1.1192399  0.69994038  0.8829070
##   17      1.0568173  0.73286887  0.8342530
##   18      1.0469036  0.73821332  0.8264794
##   19      1.0369036  0.74282876  0.8135459
##   20      1.0106942  0.75501073  0.7931272
##   21      1.0084359  0.75693801  0.7910714
##   22      1.0077410  0.75762006  0.7907431
##   23      0.9824906  0.76956926  0.7709785
##   24      0.9823339  0.76968098  0.7703055
##   25      0.9807215  0.77067359  0.7701617
##   26      0.9744918  0.77358933  0.7643884
##   27      0.9646638  0.77820852  0.7572924
##   28      0.9635618  0.77853798  0.7549437
##   29      0.9622526  0.77899192  0.7528061
##   30      0.9626567  0.77883793  0.7531227
##   31      0.9464459  0.78605250  0.7422126
##   32      0.9308610  0.79410333  0.7278662
##   33      0.9205904  0.79876222  0.7171017
##   34      0.9210315  0.79901759  0.7171882
##   35      0.9118633  0.80247076  0.7125940
##   36      0.8955050  0.80969558  0.6990453
##   37      0.8898150  0.81229196  0.6925830
##   38      0.8869713  0.81345299  0.6896258
##   39      0.8880278  0.81262668  0.6904784
##   40      0.8815142  0.81589128  0.6832985
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 40.
```

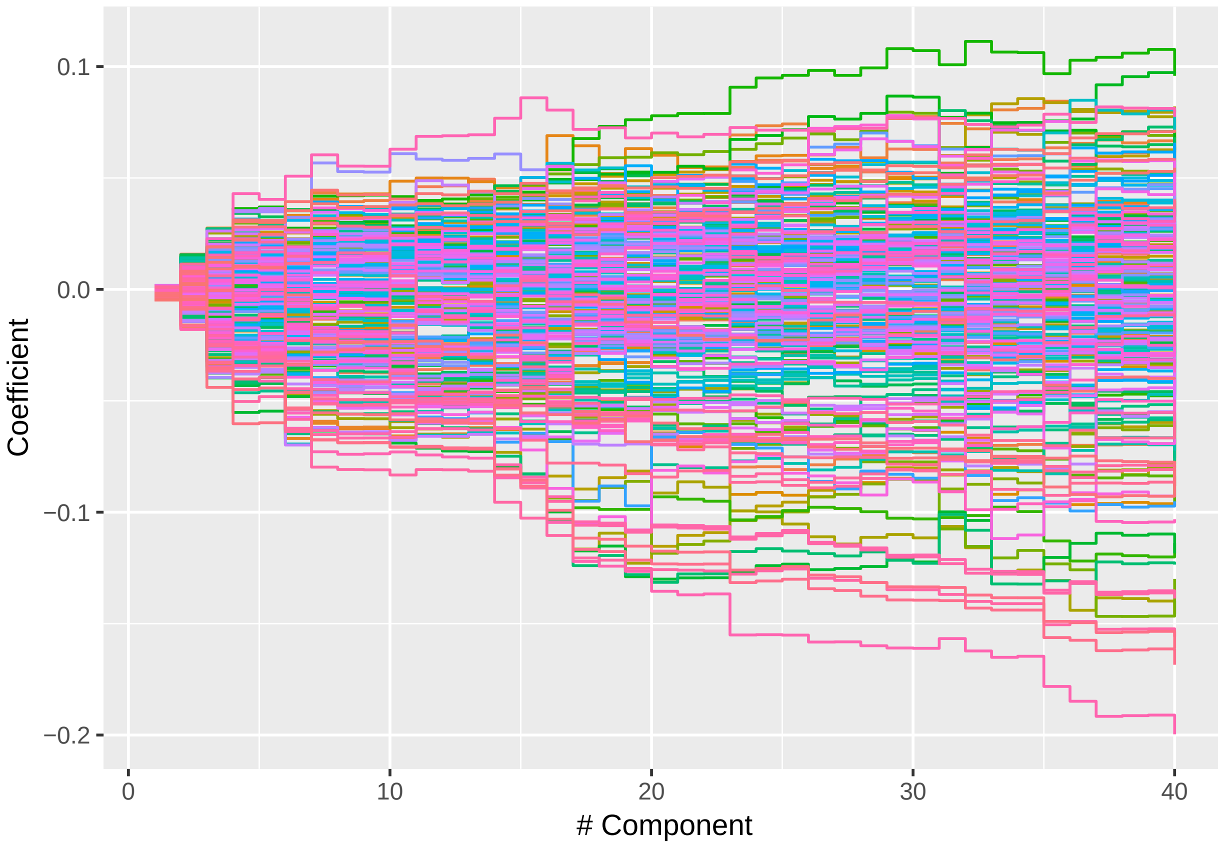
```r
# plot the RMSEs against the number of PCs
plot(pcr_fit)
```




```
# the number of PCs that gives the minimum RMSE value
pcr_fit$bestTune
```

```
##    ncomp
## 40    40
```

```
# Make a plot of the regression coefficient paths
pcr_coef <- pcr_fit$finalModel$coefficients
n <- nrow(pcr_coef)
p <- as.numeric(pcr_fit$bestTune)
variable <- rep(rownames(pcr_coef), p)
PC <- rep(1:p, each = n)
pcr_coef <- as.matrix(pcr_coef)
dat <- data.frame(variable,
                  coefficient = pcr_coef,
                  PC,
                  stringsAsFactors = FALSE)

ggplot(dat, aes(x = PC, y = coefficient, col = variable)) +
  geom_step() +
  xlab("# Component") +
  ylab("Coefficient") +
  theme(legend.position = "none")
```



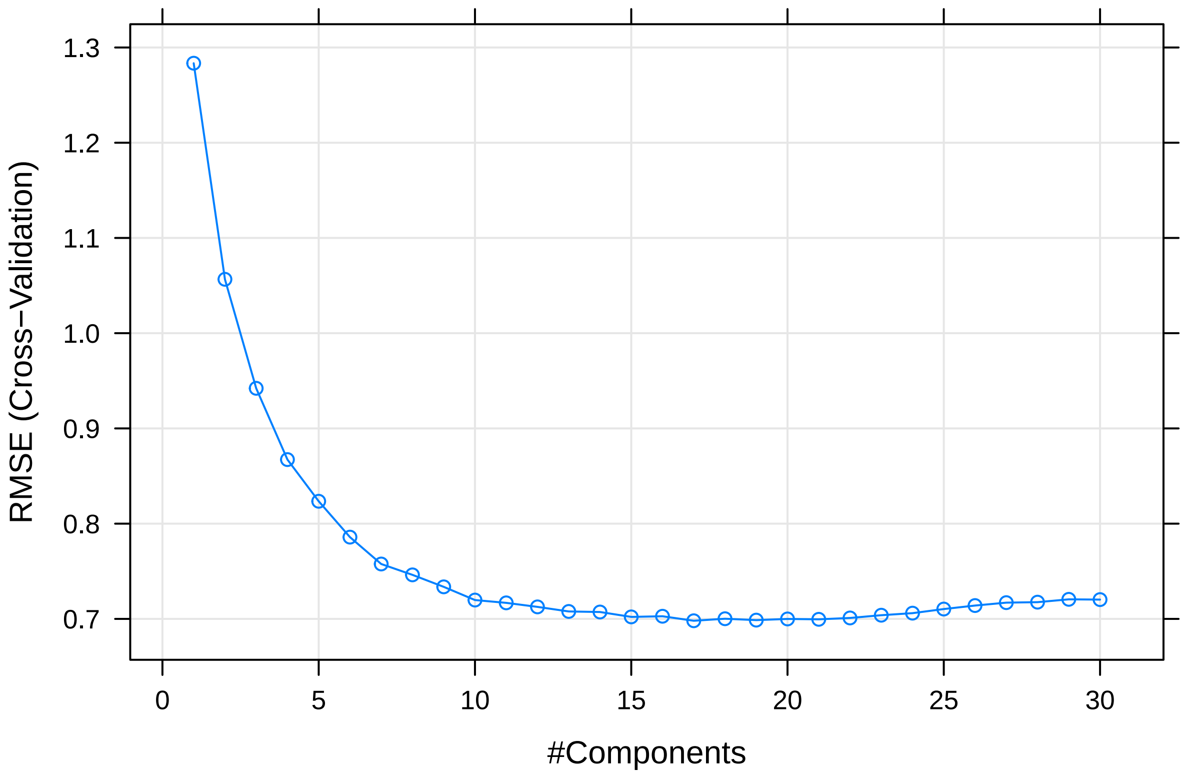
4.2) PLSR

```r
set.seed(1991)
pls_fit <- train(x = solTrainXtrans, y = solTrainY,
                 method = "pls",
                 tuneLength = 30,
                 trControl = ctrl,
                 preProcess = c("center", "scale"))

pls_fit
```

```
## Partial Least Squares
##
## 951 samples
## 228 predictors
##
## Pre-processing: centered (228), scaled (228)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 855, 857, 855, 857, 857, 855, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     1.2834799  0.6039945  0.9916002
##    2     1.0565748  0.7313375  0.8338149
##    3     0.9421131  0.7896398  0.7295970
```

```
##    4     0.8672854  0.8202404  0.6763828
##    5     0.8234981  0.8385276  0.6400779
##    6     0.7858546  0.8533009  0.6099090
##    7     0.7576326  0.8632534  0.5803275
##    8     0.7462298  0.8670690  0.5718116
##    9     0.7336407  0.8715244  0.5620758
##   10     0.7197659  0.8770720  0.5536012
##   11     0.7168387  0.8779665  0.5497677
##   12     0.7125968  0.8796722  0.5472087
##   13     0.7077955  0.8812824  0.5413748
##   14     0.7072088  0.8816105  0.5404051
##   15     0.7020926  0.8833477  0.5355129
##   16     0.7027772  0.8829757  0.5367203
##   17     0.6980264  0.8849057  0.5346358
##   18     0.7001560  0.8843897  0.5370392
##   19     0.6987062  0.8852104  0.5342642
##   20     0.6999291  0.8847274  0.5341114
##   21     0.6995291  0.8851151  0.5357248
##   22     0.7009689  0.8846909  0.5347051
##   23     0.7038482  0.8835812  0.5356718
##   24     0.7060065  0.8828946  0.5373671
##   25     0.7103322  0.8814441  0.5406134
##   26     0.7140135  0.8801809  0.5413654
##   27     0.7170543  0.8792074  0.5427062
##   28     0.7175550  0.8790947  0.5423010
##   29     0.7205393  0.8781144  0.5432047
##   30     0.7202488  0.8781004  0.5422940
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 17.
```

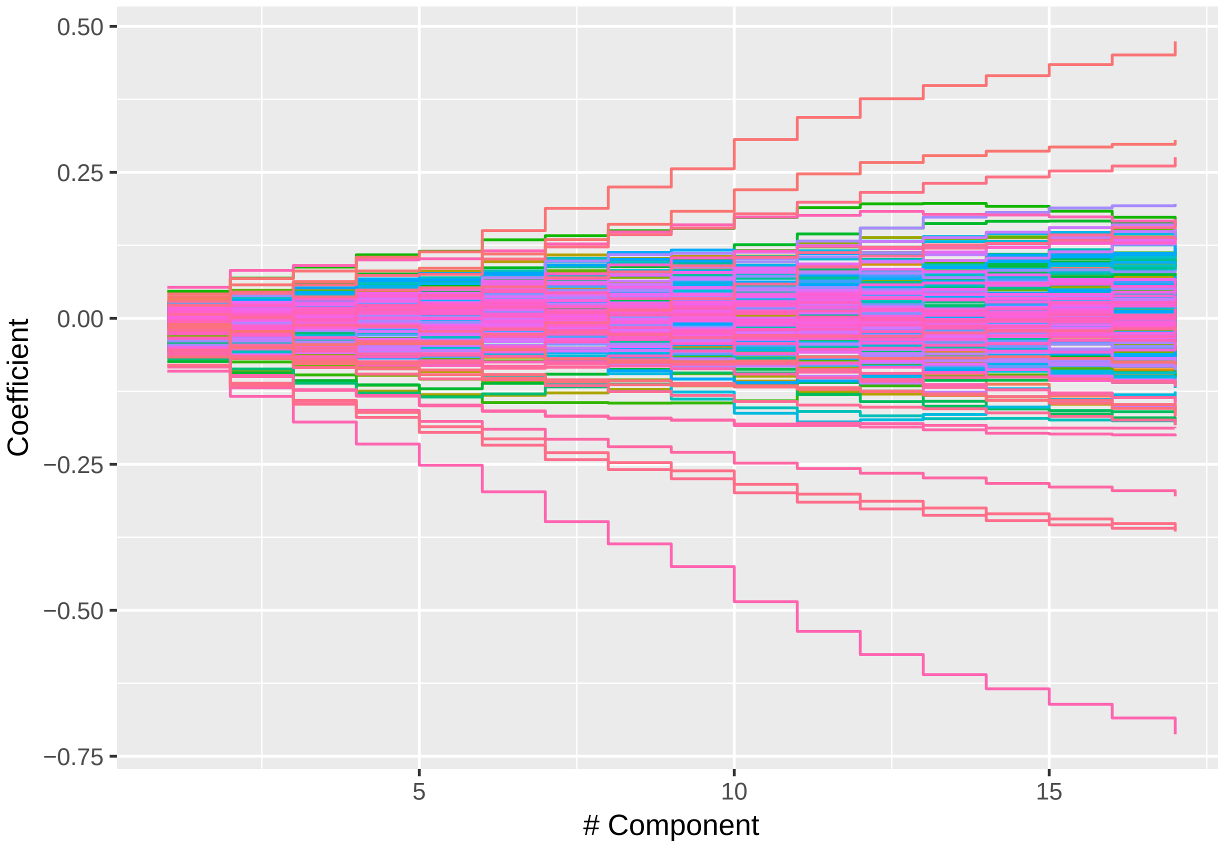
```r
# plot the RMSEs against the number of PLS components
plot(pls_fit)
```




```r
# the number of PLS components that gives the minimum RMSE value
pls_fit$bestTune
```

```
##    ncomp
## 17    17
```

```r
# make a plot of the regression coefficient paths
pls_coef <- pls_fit$finalModel$coefficients
n <- nrow(pls_coef)
p <- as.numeric(pls_fit$bestTune)
variable <- rep(rownames(pls_coef), p)
comp <- rep(1:p, each = n)
pls_coef <- as.matrix(pls_coef)

dat <- data.frame(variable,
                  coefficient = pls_coef,
                  comp)

ggplot(dat, aes(x = comp, y = coefficient, col = variable)) +
  geom_step() +
  xlab("# Component") +
  ylab("Coefficient") +
  theme(legend.position = "none")
```



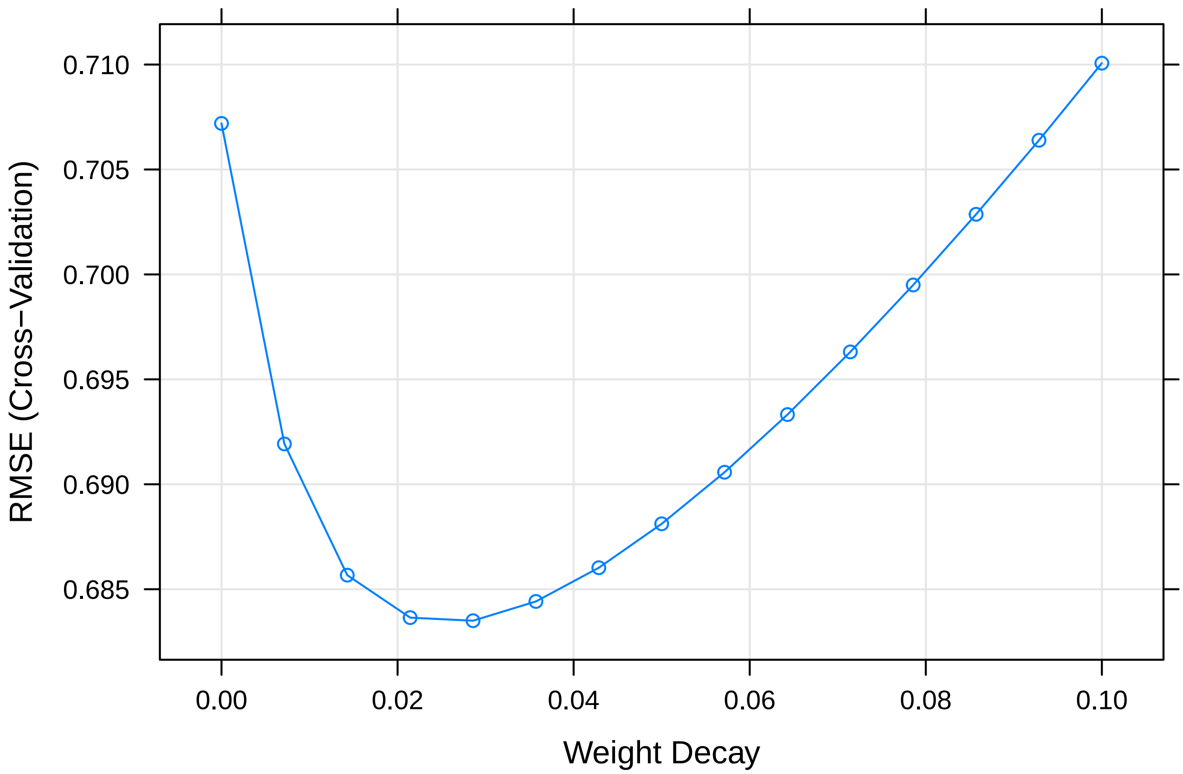
4.3) Ridge Regression

```
ridgeGrid <- data.frame(.lambda = seq(0, .1, length = 15))
set.seed(1991)
ridge_fit <- train(x = solTrainXtrans, y = solTrainY,
                   method = "ridge",
                   tuneGrid = ridgeGrid,
                   trControl = ctrl,
                   preProcess = c("center", "scale"))

ridge_fit
```

```
## Ridge Regression
##
## 951 samples
## 228 predictors
##
## Pre-processing: centered (228), scaled (228)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 855, 857, 855, 857, 857, 855, ...
## Resampling results across tuning parameters:
##
##   lambda       RMSE       Rsquared   MAE
##   0.000000000  0.7071944  0.8814257  0.5257221
##   0.007142857  0.6919235  0.8869550  0.5236807
```

```
##   0.014285714  0.6856703  0.8890874  0.5203141
##   0.021428571  0.6836460  0.8898722  0.5199234
##   0.028571429  0.6835001  0.8900898  0.5213092
##   0.035714286  0.6844180  0.8900088  0.5230327
##   0.042857143  0.6860243  0.8897518  0.5248594
##   0.050000000  0.6881167  0.8893846  0.5270827
##   0.057142857  0.6905749  0.8889460  0.5295143
##   0.064285714  0.6933224  0.8884605  0.5321482
##   0.071428571  0.6963083  0.8879441  0.5348864
##   0.078571429  0.6994975  0.8874080  0.5378075
##   0.085714286  0.7028648  0.8868596  0.5407873
##   0.092857143  0.7063920  0.8863046  0.5438502
##   0.100000000  0.7100654  0.8857467  0.5469702
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 0.02857143.
```

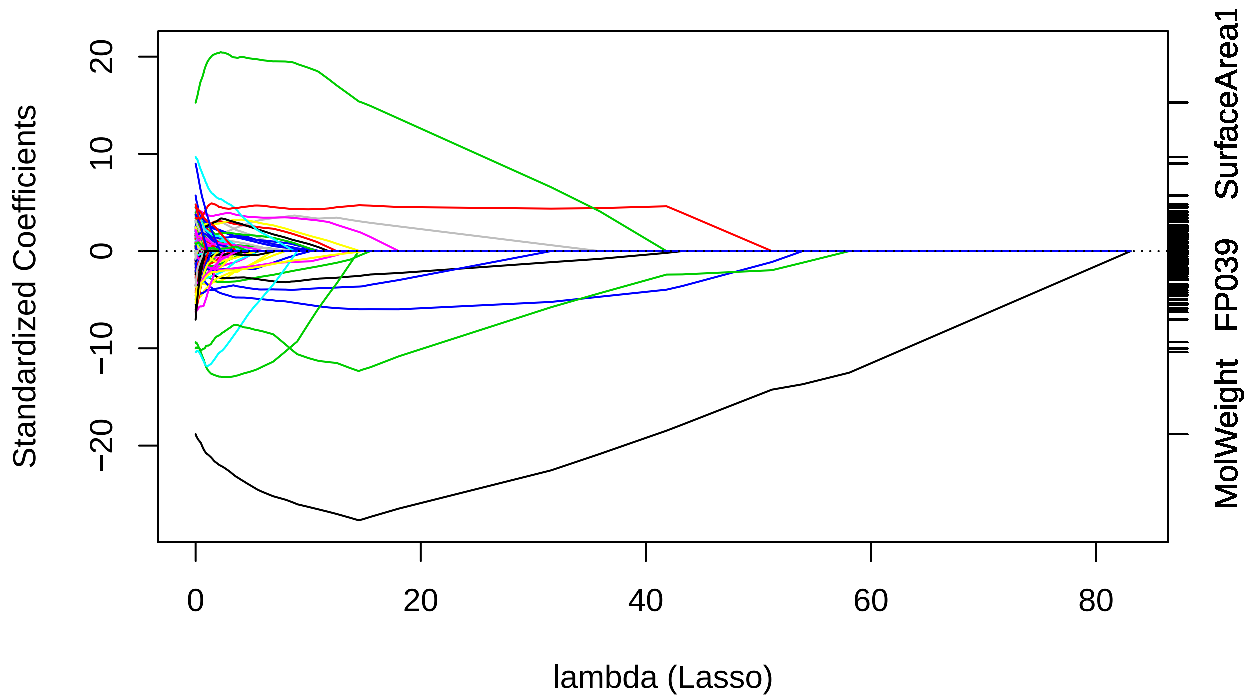
```r
# plot the RMSEs against the values of the tuning parameter lambda
plot(ridge_fit)
```




```r
# the value of lambda that gives the minimum RMSE value
ridge_fit$bestTune
```

```
##       lambda
## 5 0.02857143
```

```r
# make a plot of the regression coefficient paths
plot(ridge_fit$finalModel, xvar = "penalty", use.color = TRUE)
```



4.4) Lasso

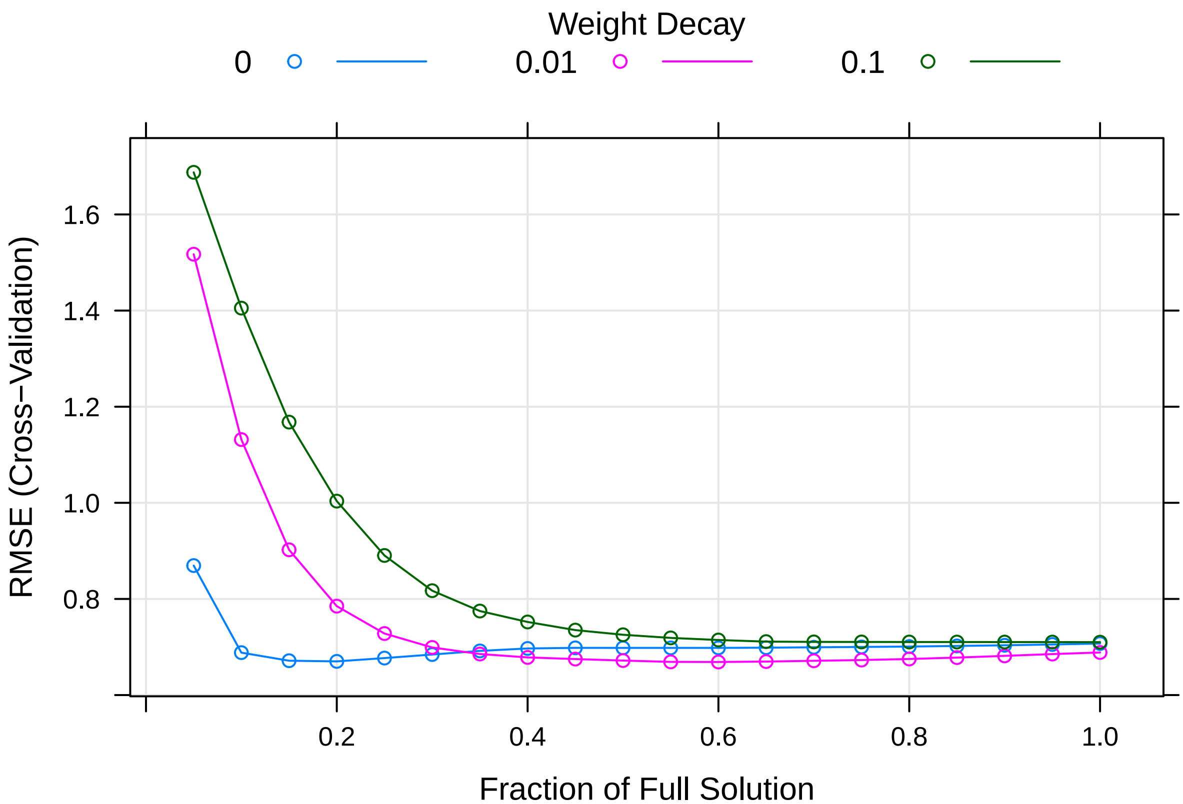
```r
enetGrid <- expand.grid(.lambda = c(0, 0.01, .1),
                        .fraction = seq(.05, 1, length = 20))
set.seed(1991)
lasso_fit <- train(x = solTrainXtrans, y = solTrainY,
                   method = "enet",
                   tuneGrid = enetGrid,
                   trControl = ctrl,
                   preProcess = c("center", "scale"))

lasso_fit
```

```
## Elasticnet
##
## 951 samples
## 228 predictors
##
## Pre-processing: centered (228), scaled (228)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 855, 857, 855, 857, 857, 855, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE       Rsquared   MAE
##   0.00    0.05      0.8694393  0.8367154  0.6593669
##   0.00    0.10      0.6882519  0.8883257  0.5249954
##   0.00    0.15      0.6714866  0.8931908  0.5133845
##   0.00    0.20      0.6701005  0.8935851  0.5125415
##   0.00    0.25      0.6769459  0.8914827  0.5126865
##   0.00    0.30      0.6843558  0.8890745  0.5179973
```

```
## 0.00 0.35 0.6918879 0.8866338 0.5234601
## 0.00 0.40 0.6970105 0.8849352 0.5280080
## 0.00 0.45 0.6981910 0.8845023 0.5288807
## 0.00 0.50 0.6981295 0.8844524 0.5284216
## 0.00 0.55 0.6981444 0.8844082 0.5278100
## 0.00 0.60 0.6981516 0.8843850 0.5269660
## 0.00 0.65 0.6985908 0.8842072 0.5259632
## 0.00 0.70 0.6994292 0.8838955 0.5254815
## 0.00 0.75 0.7001222 0.8836637 0.5251417
## 0.00 0.80 0.7009164 0.8834005 0.5248366
## 0.00 0.85 0.7021745 0.8830045 0.5247800
## 0.00 0.90 0.7035777 0.8825485 0.5248372
## 0.00 0.95 0.7052453 0.8820344 0.5251640
## 0.00 1.00 0.7071944 0.8814257 0.5257221
## 0.01 0.05 1.5172575 0.6481926 1.1629621
## 0.01 0.10 1.1316121 0.7707836 0.8644227
## 0.01 0.15 0.9022421 0.8266589 0.6836986
## 0.01 0.20 0.7849595 0.8580220 0.5983956
## 0.01 0.25 0.7280635 0.8755480 0.5555230
## 0.01 0.30 0.6990802 0.8845478 0.5341814
## 0.01 0.35 0.6854148 0.8887916 0.5246724
## 0.01 0.40 0.6783782 0.8909363 0.5192493
## 0.01 0.45 0.6749279 0.8919645 0.5173129
## 0.01 0.50 0.6717883 0.8929277 0.5154451
## 0.01 0.55 0.6690558 0.8938298 0.5135199
## 0.01 0.60 0.6687896 0.8939771 0.5131305
## 0.01 0.65 0.6697231 0.8937514 0.5126997
## 0.01 0.70 0.6712940 0.8933490 0.5123954
## 0.01 0.75 0.6728448 0.8928764 0.5123212
## 0.01 0.80 0.6750877 0.8921643 0.5130300
## 0.01 0.85 0.6779499 0.8912939 0.5144052
## 0.01 0.90 0.6815677 0.8901898 0.5167259
## 0.01 0.95 0.6852845 0.8890690 0.5194156
## 0.01 1.00 0.6886007 0.8880743 0.5217469
## 0.10 0.05 1.6876421 0.5172217 1.2938453
## 0.10 0.10 1.4050887 0.7004963 1.0731807
## 0.10 0.15 1.1679369 0.7633412 0.8905843
## 0.10 0.20 1.0035064 0.7912991 0.7622082
## 0.10 0.25 0.8904595 0.8233943 0.6744600
## 0.10 0.30 0.8172256 0.8436821 0.6226784
## 0.10 0.35 0.7748367 0.8565541 0.5963412
## 0.10 0.40 0.7521196 0.8648159 0.5780088
## 0.10 0.45 0.7352072 0.8716584 0.5647150
## 0.10 0.50 0.7254667 0.8759264 0.5577237
## 0.10 0.55 0.7189654 0.8789894 0.5534675
## 0.10 0.60 0.7145841 0.8810958 0.5507771
## 0.10 0.65 0.7112406 0.8827524 0.5481277
## 0.10 0.70 0.7106350 0.8834483 0.5473125
## 0.10 0.75 0.7105176 0.8838941 0.5467777
## 0.10 0.80 0.7103955 0.8843254 0.5466359
## 0.10 0.85 0.7103572 0.8847166 0.5469637
## 0.10 0.90 0.7103120 0.8850839 0.5471206
## 0.10 0.95 0.7102830 0.8854133 0.5471128
## 0.10 1.00 0.7100654 0.8857467 0.5469702
```

```
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were fraction = 0.6 and lambda = 0.01.
```

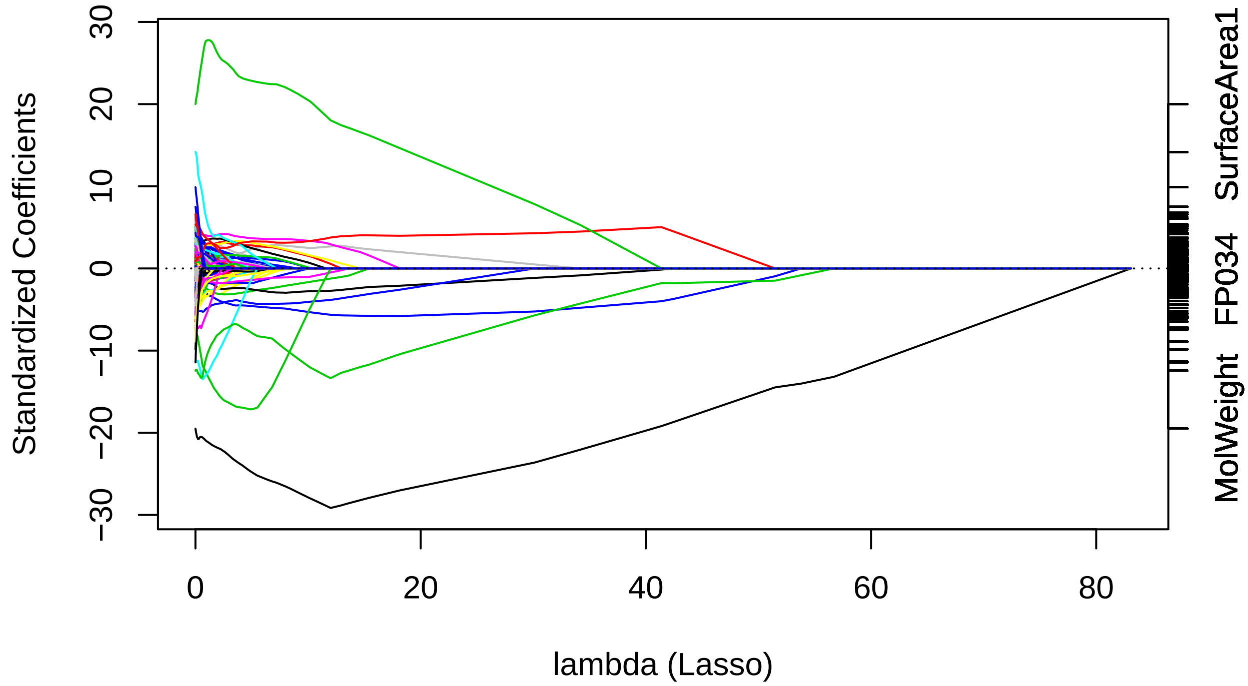
```r
# plot the RMSEs against the values of the tuning parameter lambda
plot(lasso_fit)
```




```r
# the value of lambda that gives the minimum RMSE value
lasso_fit$bestTune
```

```
##    fraction lambda
## 32      0.6   0.01
```

```r
# make a plot of the regression coefficient paths
plot(lasso_fit$finalModel, xvar = "penalty", use.color = TRUE)
```



4.5) Model Selection

```
# compute test-MSEs for PCR
y_pcr <- predict(pcr_fit, solTestXtrans)
MSE_pcr <- mean((solTestY - y_pcr)^2)
MSE_pcr
```

```
## [1] 0.8070032
```

```
# compute test-MSEs for PLSR
y_pls <- predict(pls_fit, solTestXtrans)
MSE_pls <- mean((solTestY - y_pls)^2)
MSE_pls
```

```
## [1] 0.5326949
```

```
# compute test-MSEs for Ridge Regression
y_ridge <- predict(ridge_fit, solTestXtrans)
MSE_ridge <- mean((solTestY - y_ridge)^2)
MSE_ridge
```

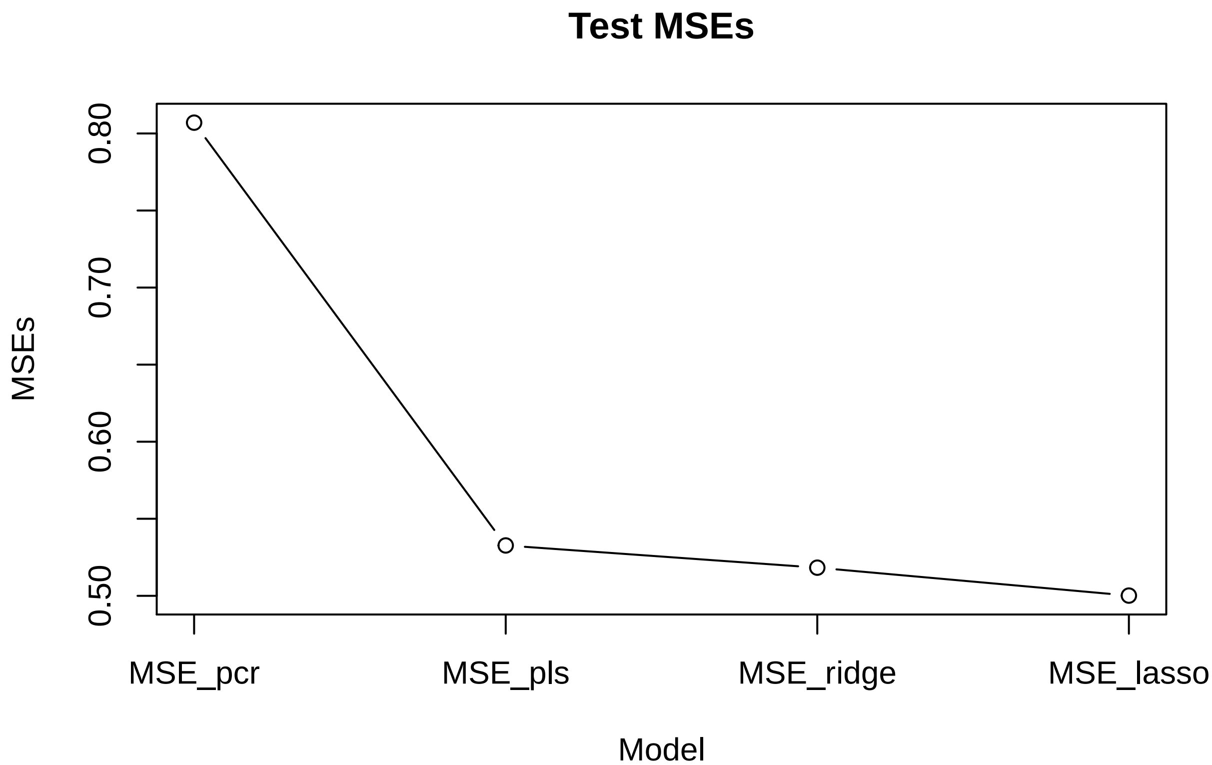
```
## [1] 0.5182821
```

```
# compute test-MSEs for PCR
y_lasso <- predict(lasso_fit, solTestXtrans)
MSE_lasso <- mean((solTestY - y_lasso)^2)
MSE_lasso
```

```
## [1] 0.5001638
```

```
# graph the test-MSEs
MSEs <- rbind(MSE_pcr, MSE_pls, MSE_ridge, MSE_lasso)
plot(MSEs, main = "Test MSEs", xlab = "Model", type = "b", xaxt = "n")
```

```r
axis(1, at = 1:4, labels = rownames(MSEs))
```




```r
# the smallest test-MSE
MSEs[which.min(MSEs), 1]
```

```
## MSE_lasso
## 0.5001638
```