PREDICT422-Programming Assignment 3: Resampling, Model Selection, and Regularization

Artur Mrozowski

April 26, 2017

Programming Assignment 3: Resampling, Model Selection, and Regularization ############################
#Exercise 5 5.4 ISLR ##

```r
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.2.3
```

```r
attach(Default)
```

(a) Fit a logistic regression model that uses income and balance to predict default.

```r
set.seed(1)
glm.fit = glm(default ~ income + balance, data = Default, family = binomial)
```

(b) Using the validation set approach, estimate the test error of this model. In order to do this, you must perform the following steps:

```r
fiveB = function() {
#(b) i. Split the sample set into a training set and a validation set.
#train= sample(10000,7000)
train=sample(dim(Default)[1],dim(Default)[1]*0.5)
#test=Default[)-train]

##ii. Fit a multiple logistic regression model using only the training
##observations.
glm.fit = glm(default ~ income + balance, data = Default, family = binomial,
              subset = train)

##iii. Obtain a prediction of default status for each individual in
#the validation set by computing the posterior probability of
#default for that individual, and classifying the individual to
#the default category if the posterior probability is greater
#than 0.5.

# iii.
glm.pred = rep("No", dim(Default)[1]/2)
glm.probs = predict(glm.fit, Default[-train, ], type = "response")
glm.pred[glm.probs > 0.5] = "Yes"
##iv
##Compute the validation set error, which is the fraction of
#the observations in the validation set that are misclassified.
return (mean(glm.pred != Default[-train, ]$default))
}
fiveB()
```

```
## [1] 0.0286
```

(c) Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.

```
for (i in 1:3)
  print(fiveB())
```

```
## [1] 0.0236
## [1] 0.028
## [1] 0.0268
```

It seems to average around 2.6% test error rate

(d) Now consider a logistic regression model that predicts the probability of default using income, balance, and a dummy variable for student. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for student leads to a reduction in the test error rate.

```
train = sample(dim(Default)[1], dim(Default)[1]/2)
glm.fit = glm(default ~ income + balance + student, data = Default, family = binomial,
              subset = train)
glm.pred = rep("No", dim(Default)[1]/2)
glm.probs = predict(glm.fit, Default[-train, ], type = "response")
glm.pred[glm.probs > 0.5] = "Yes"
mean(glm.pred != Default[-train, ]$default)
```

```
## [1] 0.0264
```

2.64% test error rate, with student dummy variable. Using the validation set approach, it doesn't appear adding the student dummy variable leads to a reduction in the test error rate.

8. We will now perform cross-validation on a simulated data set.

(a) Generate a simulated data set as follows: In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

```
set.seed(1)
y = rnorm(100)
x = rnorm(100)
y = x - 2 * x^2 + rnorm(100)
```

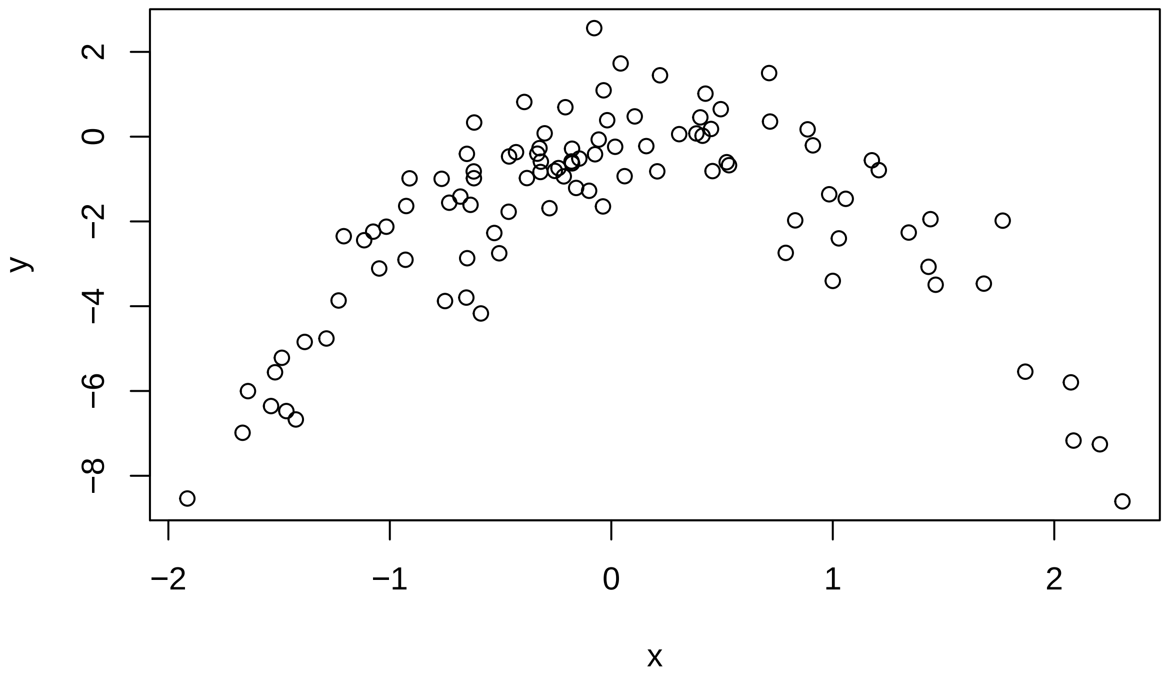
n = 100, p = 2.

Y=X???2X^2+??

Y=X-2x^2+??

(b) Create a scatterplot of X against Y . Comment on what you find.

```r
plot(x, y)
```



Quadratic plot

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

i.Y = B0 + B1x + e

```r
library(boot)
```

```
## Warning: package 'boot' was built under R version 3.2.2
```

```r
Data = data.frame(x, y)
set.seed(1)
# i.
glm.fit = glm(y ~ x)
cv.glm(Data, glm.fit)$delta
```

```
## [1] 5.890979 5.888812
```

ii. Y= B0 + B1x + B2x2 +e

```r
# ii.
glm.fit = glm(y ~ poly(x, 2))
cv.glm(Data, glm.fit)$delta
```

```
## [1] 1.086596 1.086326
```

iii. Y= B0 + B1x + B2x2 +b3x3 +e

```r
# iii.
glm.fit = glm(y ~ poly(x, 3))
cv.glm(Data, glm.fit)$delta
```

```
## [1] 1.102585 1.102227
```

iv. Y= B0 + B1x + B2x2 +b3x3+ b4x4 +e

```r
# iv.
glm.fit = glm(y ~ poly(x, 4))
cv.glm(Data, glm.fit)$delta
```

```
## [1] 1.114772 1.114334
```

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```r
set.seed(10)
# i.
glm.fit = glm(y ~ x)
cv.glm(Data, glm.fit)$delta
```

```
## [1] 5.890979 5.888812
```

```r
# ii.
glm.fit = glm(y ~ poly(x, 2))
cv.glm(Data, glm.fit)$delta
```

```
## [1] 1.086596 1.086326
```

```r
# iii.
glm.fit = glm(y ~ poly(x, 3))
cv.glm(Data, glm.fit)$delta
```

```
## [1] 1.102585 1.102227
```

```r
# iv.
glm.fit = glm(y ~ poly(x, 4))
cv.glm(Data, glm.fit)$delta
```

```
## [1] 1.114772 1.114334
```

Exact same, because LOOCV will be the same since it evaluates n folds of a single observation.

(e) Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

The quadratic polynomial had the lowest LOOCV test error rate. This was expected because it matches the true form of Y.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

```r
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x, 4))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8914  -0.5244   0.0749   0.5932   2.7796
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277     0.1041 -17.549   <2e-16 ***
## poly(x, 4)1   2.3164     1.0415   2.224   0.0285 *
## poly(x, 4)2 -21.0586     1.0415 -20.220   <2e-16 ***
## poly(x, 4)3  -0.3048     1.0415  -0.293   0.7704
## poly(x, 4)4  -0.4926     1.0415  -0.473   0.6373
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.084654)
##
##     Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.04  on 95  degrees of freedom
## AIC: 298.78
##
## Number of Fisher Scoring iterations: 2
```

p-values show statistical significance of linear and quadratic terms, which agrees with the CV results.

Excercise 8 ISLR 6.8

(a) Use the rnorm() function to generate a predictor X of length n = 100, as well as a noise vector e of length n = 100.

```
set.seed(1)
X = rnorm(100)
eps = rnorm(100)
```

(b) Generate a response vector Y of length n = 100 according to the model Y = ??0 + ??1X + ??2X2 + ??3X3 + e, where ??0, ??1, ??2, and ??3 are constants of your choice.

```
beta0 = 3
beta1 = 2
beta2 = -3
beta3 = 0.3
Y = beta0 + beta1 * X + beta2 * X^2 + beta3 * X^3 + eps
```

(c) Use the regsubsets() function to perform best subset selection in order to choose the best model containing the predictors X,X2, . . .,X10. What is the best model obtained according to Cp, BIC, and adjusted R2? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained. Note you will need to use the data.frame() function to create a single data set containing both X and Y .

```
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 3.2.5
```

```
data.full = data.frame(y = Y, x = X)
mod.full = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10)
mod.summary = summary(mod.full)

# Find the model size for best cp, BIC and adjr2
which.min(mod.summary$cp)
```

```
## [1] 3
```

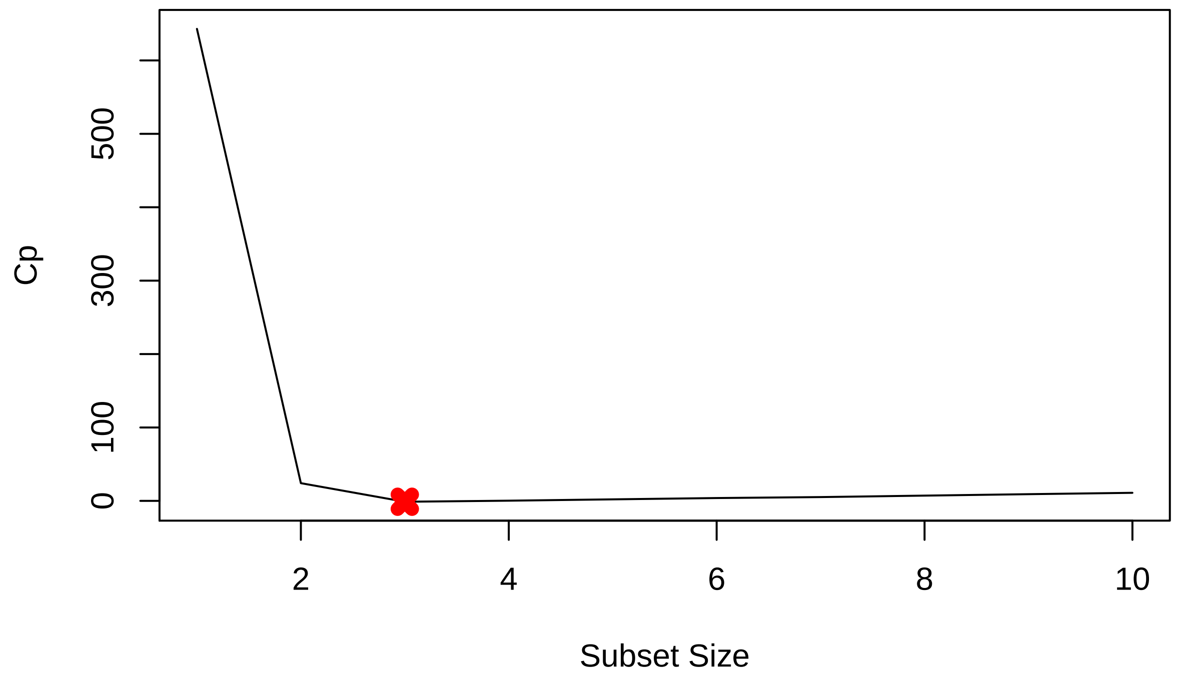
```
which.min(mod.summary$bic)
```

```
## [1] 3
```

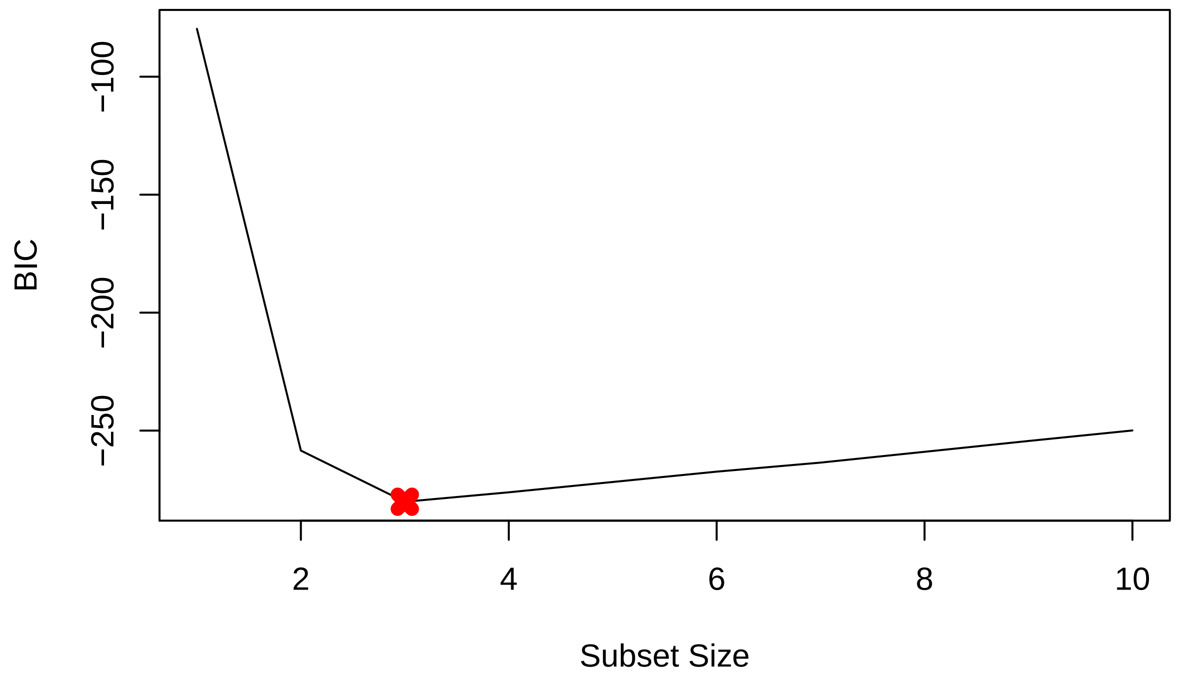
```
which.max(mod.summary$adjr2)
```

```
## [1] 3
```

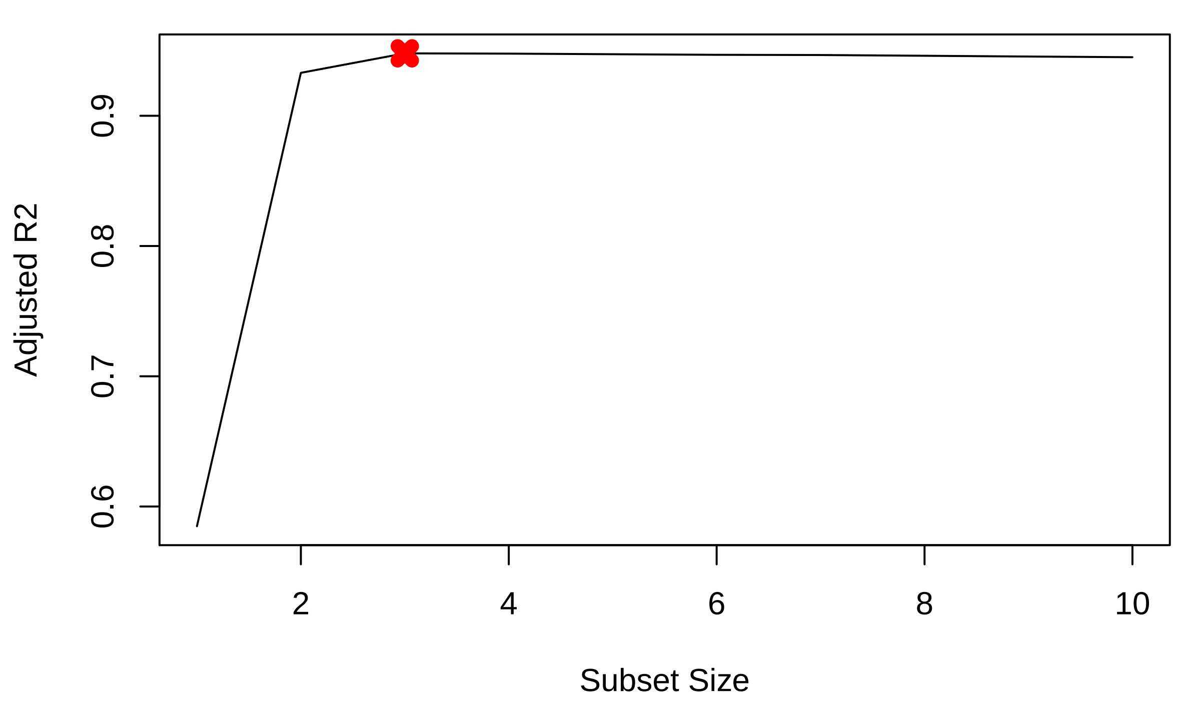
```
# Plot cp, BIC and adjr2
plot(mod.summary$cp, xlab = "Subset Size", ylab = "Cp", pch = 20, type = "l")
points(3, mod.summary$cp[3], pch = 4, col = "red", lwd = 7)
```



```r
plot(mod.summary$bic, xlab = "Subset Size", ylab = "BIC", pch = 20, type = "l")
points(3, mod.summary$bic[3], pch = 4, col = "red", lwd = 7)
```



```r
plot(mod.summary$adjr2, xlab = "Subset Size", ylab = "Adjusted R2", pch = 20,
    type = "l")
points(3, mod.summary$adjr2[3], pch = 4, col = "red", lwd = 7)
```



```r
coefficients(mod.full, id = 3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2 
##            3.07627412            2.35623596           -3.16514887 
## poly(x, 10, raw = T)7 
##            0.01046843
```

All statistics pick X7 over X3. The remaining coefficients are quite close to ???? s.

(d) Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)?

```r
mod.fwd = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10,
    method = "forward")
mod.bwd = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10,
    method = "backward")
fwd.summary = summary(mod.fwd)
bwd.summary = summary(mod.bwd)
which.min(fwd.summary$cp)
```

```
## [1] 3
```

```r
which.min(bwd.summary$cp)
```

```
## [1] 3
```

```r
which.min(fwd.summary$bic)
```

```
## [1] 3
```

```r
which.min(bwd.summary$bic)
```

```
## [1] 3
```

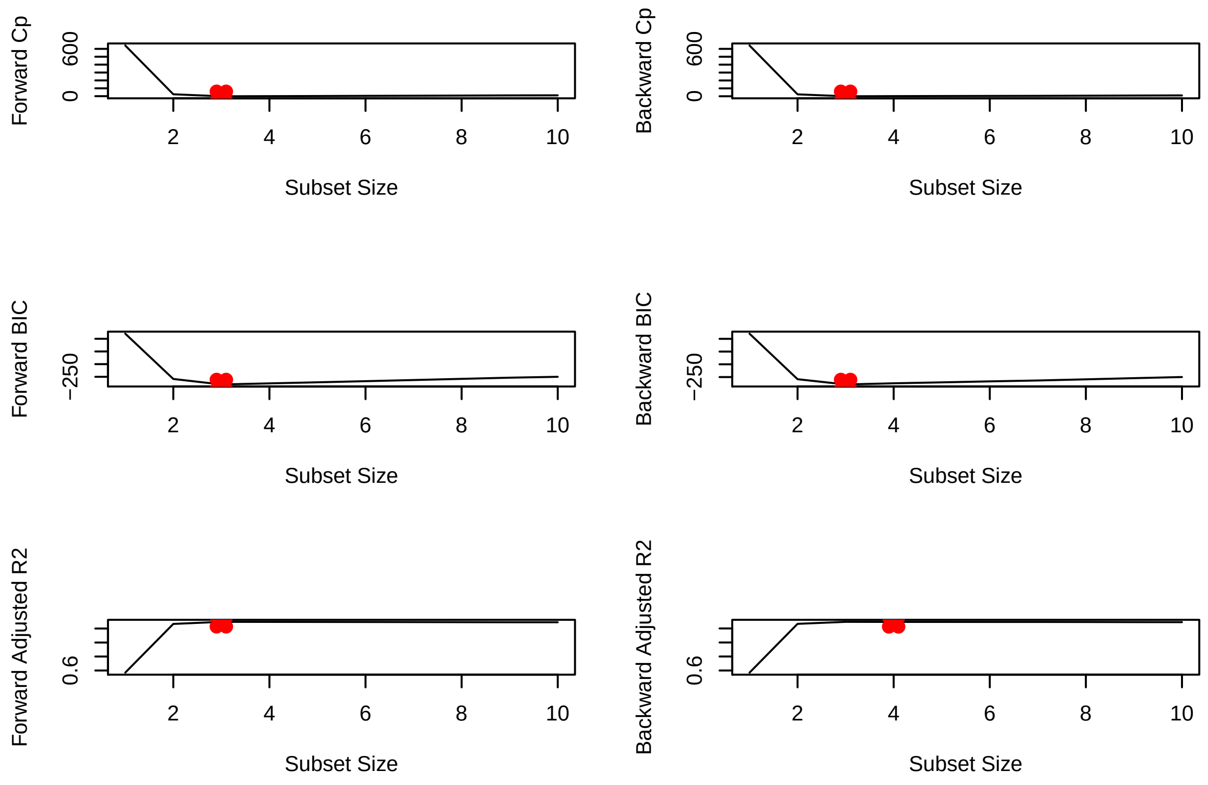
```r
which.max(fwd.summary$adjr2)
```

```
## [1] 3
```

```r
which.max(bwd.summary$adjr2)
```

```
## [1] 3
```

```r
# Plot the statistics
par(mfrow = c(3, 2))
plot(fwd.summary$cp, xlab = "Subset Size", ylab = "Forward Cp", pch = 20, type = "l")
points(3, fwd.summary$cp[3], pch = 4, col = "red", lwd = 7)
plot(bwd.summary$cp, xlab = "Subset Size", ylab = "Backward Cp", pch = 20, type = "l")
points(3, bwd.summary$cp[3], pch = 4, col = "red", lwd = 7)
plot(fwd.summary$bic, xlab = "Subset Size", ylab = "Forward BIC", pch = 20,
    type = "l")
points(3, fwd.summary$bic[3], pch = 4, col = "red", lwd = 7)
plot(bwd.summary$bic, xlab = "Subset Size", ylab = "Backward BIC", pch = 20,
    type = "l")
points(3, bwd.summary$bic[3], pch = 4, col = "red", lwd = 7)
plot(fwd.summary$adjr2, xlab = "Subset Size", ylab = "Forward Adjusted R2",
    pch = 20, type = "l")
points(3, fwd.summary$adjr2[3], pch = 4, col = "red", lwd = 7)
plot(bwd.summary$adjr2, xlab = "Subset Size", ylab = "Backward Adjusted R2",
    pch = 20, type = "l")
points(4, bwd.summary$adjr2[4], pch = 4, col = "red", lwd = 7)
```



We see that all statistics pick 33 variable models except backward stepwise with adjusted R2. Here are the coefficients

```r
coefficients(mod.fwd, id = 3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##            3.07627412            2.35623596           -3.16514887
## poly(x, 10, raw = T)7
##            0.01046843
```

```r
coefficients(mod.bwd, id = 3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.078881355           2.419817953          -3.177235617
## poly(x, 10, raw = T)9
##           0.001870457
```

```r
coefficients(mod.fwd, id = 4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.112358625           2.369858879          -3.275726574
## poly(x, 10, raw = T)4 poly(x, 10, raw = T)7
##           0.027673638           0.009997134
```

Here forward stepwise picks X7X7 over X3X3. Backward stepwise with 33 variables picks X9X9 while backward stepwise with 44 variables picks X4X4 and X7X7. All other coefficients are close to ???? s.

(e) Now fit a lasso model to the simulated data, again using X,X2, . . . , X10 as predictors. Use cross-validation to select the optimal value of ??. Create plots of the cross-validation error as a function of ??. Report the resulting coefficient estimates, and discuss the results obtained.

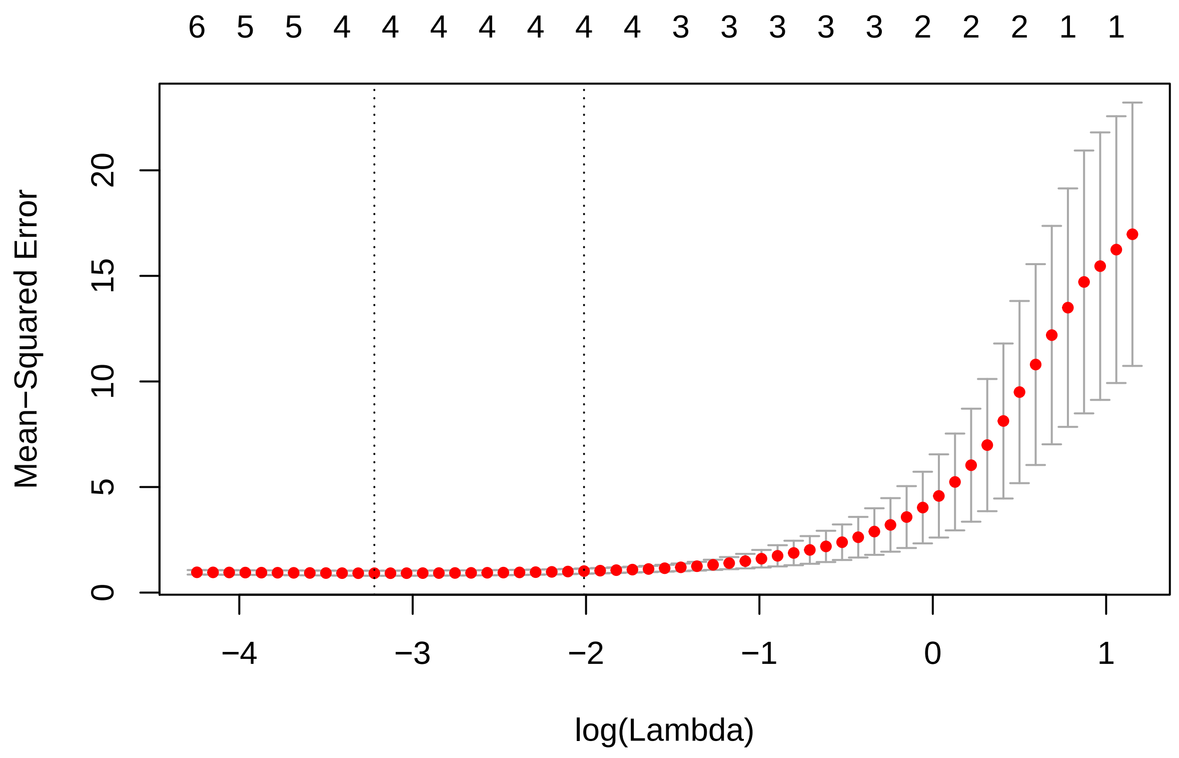
```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.2.5

## Loading required package: Matrix

## Loading required package: foreach

## Warning: package 'foreach' was built under R version 3.2.3

## Loaded glmnet 2.0-5
```

```
xmat = model.matrix(y ~ poly(x, 10, raw = T), data = data.full)[, -1]
mod.lasso = cv.glmnet(xmat, Y, alpha = 1)
best.lambda = mod.lasso$lambda.min
best.lambda
```

```
## [1] 0.03991416
```

```
plot(mod.lasso)
```



```r
# Next fit the model on entire data using best lambda
best.model = glmnet(xmat, Y, alpha = 1)
predict(best.model, s = best.lambda, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                   1
## (Intercept)              3.0398151056
## poly(x, 10, raw = T)1    2.2303371338
## poly(x, 10, raw = T)2   -3.1033192679
## poly(x, 10, raw = T)3    .
## poly(x, 10, raw = T)4    .
## poly(x, 10, raw = T)5    0.0498410763
## poly(x, 10, raw = T)6    .
## poly(x, 10, raw = T)7    0.0008068431
## poly(x, 10, raw = T)8    .
## poly(x, 10, raw = T)9    .
## poly(x, 10, raw = T)10   .
```

Lasso also picks X5 over X3. It also picks X7X7 with negligible coefficient.

(f) Now generate a response vector Y according to the model Y = ??0 + ??7X7 + e, and perform best subset selection and the lasso. Discuss the results obtained.

```r
beta7 = 7
Y = beta0 + beta7 * X^7 + eps
# Predict using regsubsets
data.full = data.frame(y = Y, x = X)
mod.full = regsubsets(y ~ poly(x, 10, raw = T), data = data.full, nvmax = 10)
mod.summary = summary(mod.full)

# Find the model size for best cp, BIC and adjr2
which.min(mod.summary$cp)
```

```
## [1] 2
```

```r
which.min(mod.summary$bic)
```

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

```r
which.max(mod.summary$adjr2)
```

```
## [1] 4
```

```r
coefficients(mod.full, id = 1)
```

```
##          (Intercept) poly(x, 10, raw = T)7
##              2.95894               7.00077
```

```r
coefficients(mod.full, id = 2)
```

```
##          (Intercept) poly(x, 10, raw = T)2 poly(x, 10, raw = T)7
##            3.0704904            -0.1417084             7.0015552
```

```r
coefficients(mod.full, id = 4)
```

```
##          (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##            3.0762524             0.2914016            -0.1617671
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)7
##           -0.2526527             7.0091338
```

We see that BIC picks the most accurate 1-variable model with matching coefficients. Other criteria pick additional variables.

```r
xmat = model.matrix(y ~ poly(x, 10, raw = T), data = data.full)[, -1]
mod.lasso = cv.glmnet(xmat, Y, alpha = 1)
best.lambda = mod.lasso$lambda.min
best.lambda
```

```
## [1] 13.57478
```

```r
best.model = glmnet(xmat, Y, alpha = 1)
predict(best.model, s = best.lambda, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)            3.904188
## poly(x, 10, raw = T)1  .
## poly(x, 10, raw = T)2  .
## poly(x, 10, raw = T)3  .
## poly(x, 10, raw = T)4  .
## poly(x, 10, raw = T)5  .
## poly(x, 10, raw = T)6  .
## poly(x, 10, raw = T)7  6.776797
## poly(x, 10, raw = T)8  .
## poly(x, 10, raw = T)9  .
## poly(x, 10, raw = T)10 .
```

Lasso also picks the best 1-variable model but intercept is quite off (3.83.8 vs 33).

Excercise 9 (a) Split the data set into a training set and a test set.

```r
library(ISLR)
set.seed(11)
sum(is.na(College))
```

```
## [1] 0
```

```r
train.size = dim(College)[1] / 2
train = sample(1:dim(College)[1], train.size)
test = -train
College.train = College[train, ]
College.test = College[test, ]
```

(b) Fit a linear model using least squares on the training set, and report the test error obtained.

```r
lm.fit = lm(Apps~., data=College.train)
lm.pred = predict(lm.fit, College.test)
mean((College.test[, "Apps"] - lm.pred)^2)
```

```
## [1] 1538442
```

Test RSS is 1538442

(c) Fit a ridge regression model on the training set, with ?? chosen by cross-validation. Report the test error obtained.

```r
library(glmnet)
```

```r
train.mat = model.matrix(Apps~., data=College.train)
test.mat = model.matrix(Apps~., data=College.test)
grid = 10 ^ seq(4, -2, length=100)
mod.ridge = cv.glmnet(train.mat, College.train[, "Apps"], alpha=0, lambda=grid, thresh=1e-12)
lambda.best = mod.ridge$lambda.min
lambda.best
```

```
## [1] 18.73817
```

```r
ridge.pred = predict(mod.ridge, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - ridge.pred)^2)
```

```
## [1] 1608859
```

Test RSS is slightly higher that OLS, 16088591608859.

(d) Fit a lasso model on the training set, with ?? chosen by crossvalidation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```r
mod.lasso = cv.glmnet(train.mat, College.train[, "Apps"], alpha=1, lambda=grid, thresh=1e-12)
lambda.best = mod.lasso$lambda.min
lambda.best
```

```
## [1] 21.54435
```

```
lasso.pred = predict(mod.lasso, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - lasso.pred)^2)
```

```
## [1] 1635280
```

Again, Test RSS is slightly higher that OLS, 16352801635280.

The coefficients look like

```
mod.lasso = glmnet(model.matrix(Apps~., data=College), College[, "Apps"], alpha=1)
predict(mod.lasso, s=lambda.best, type="coefficients")
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) -6.038452e+02
## (Intercept)  .
## PrivateYes  -4.235413e+02
## Accept       1.455236e+00
## Enroll      -2.003696e-01
## Top10perc    3.367640e+01
## Top25perc   -2.403036e+00
## F.Undergrad  .
## P.Undergrad  2.086035e-02
## Outstate    -5.781855e-02
## Room.Board   1.246462e-01
## Books        .
## Personal     1.832912e-05
## PhD         -5.601313e+00
## Terminal    -3.313824e+00
## S.F.Ratio    4.478684e+00
## perc.alumni -9.796600e-01
## Expend       6.967693e-02
## Grad.Rate    5.159652e+00
```

(e) Fit a PCR model on the training set, with M chosen by crossvalidation. Report the test error obtained, along with the value of M selected by cross-validation.

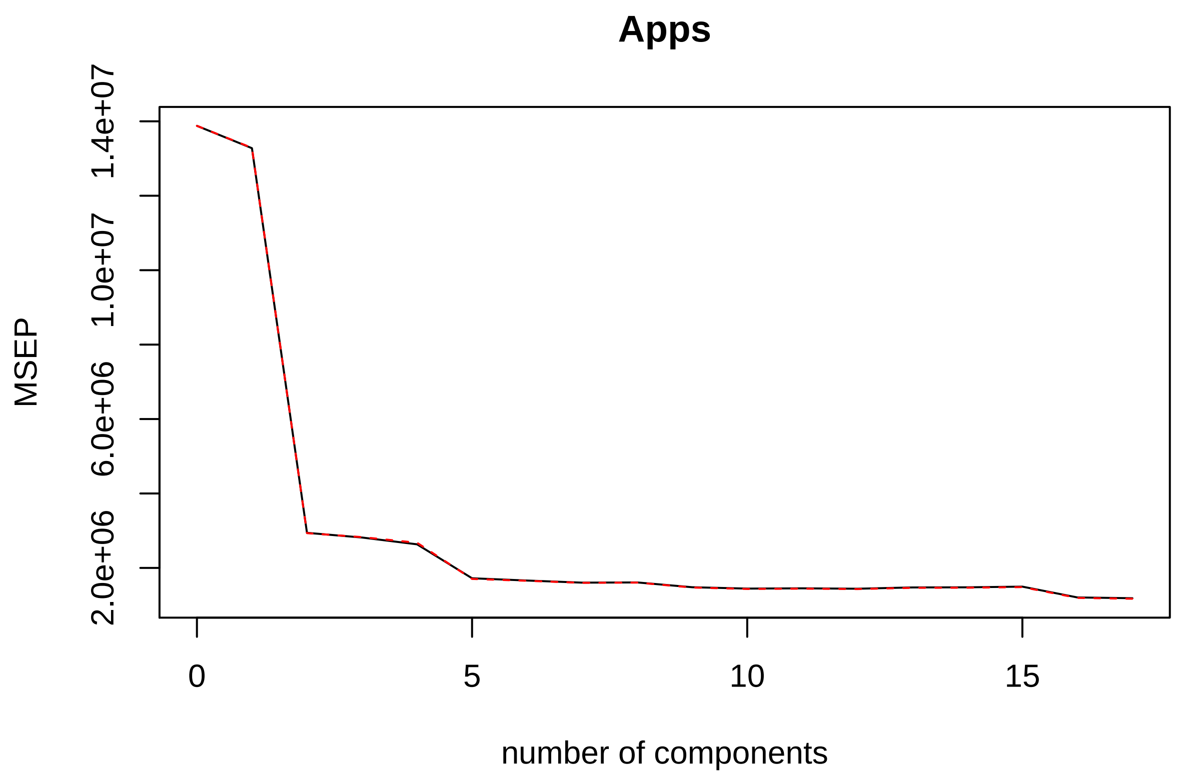
```
library(pls)
```

```
## Warning: package 'pls' was built under R version 3.2.3
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
pcr.fit = pcr(Apps~., data=College.train, scale=T, validation="CV")
validationplot(pcr.fit, val.type="MSEP")
```



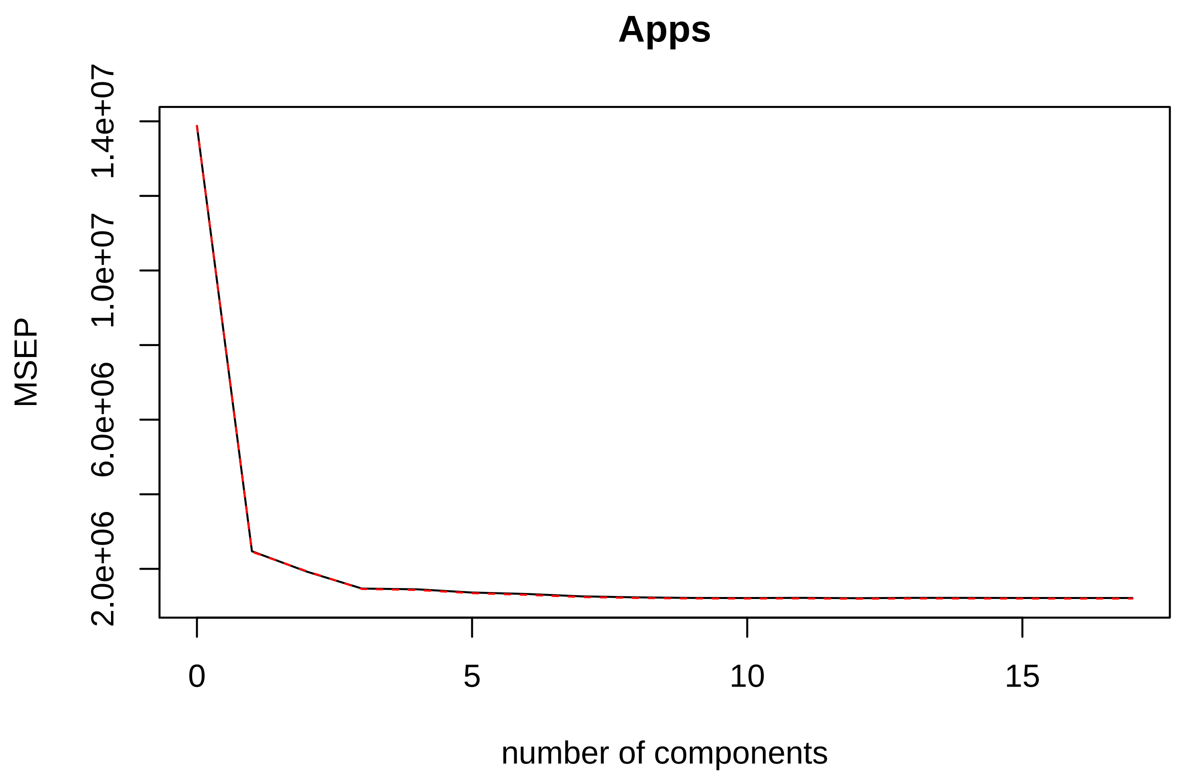
```r
pcr.pred = predict(pcr.fit, College.test, ncomp=10)
mean((College.test[, "Apps"] - data.frame(pcr.pred))^2)
```

```
## [1] 3014496
```

Test RSS for PCR is about 30144963014496.

(f) Fit a PLS model on the training set, with M chosen by crossvalidation. Report the test error obtained, along with the value of M selected by cross-validation.

```r
pls.fit = plsr(Apps~., data=College.train, scale=T, validation="CV")
validationplot(pls.fit, val.type="MSEP")
```



```
pls.pred = predict(pls.fit, College.test, ncomp=10)
mean((College.test[, "Apps"] - data.frame(pls.pred))^2)
```

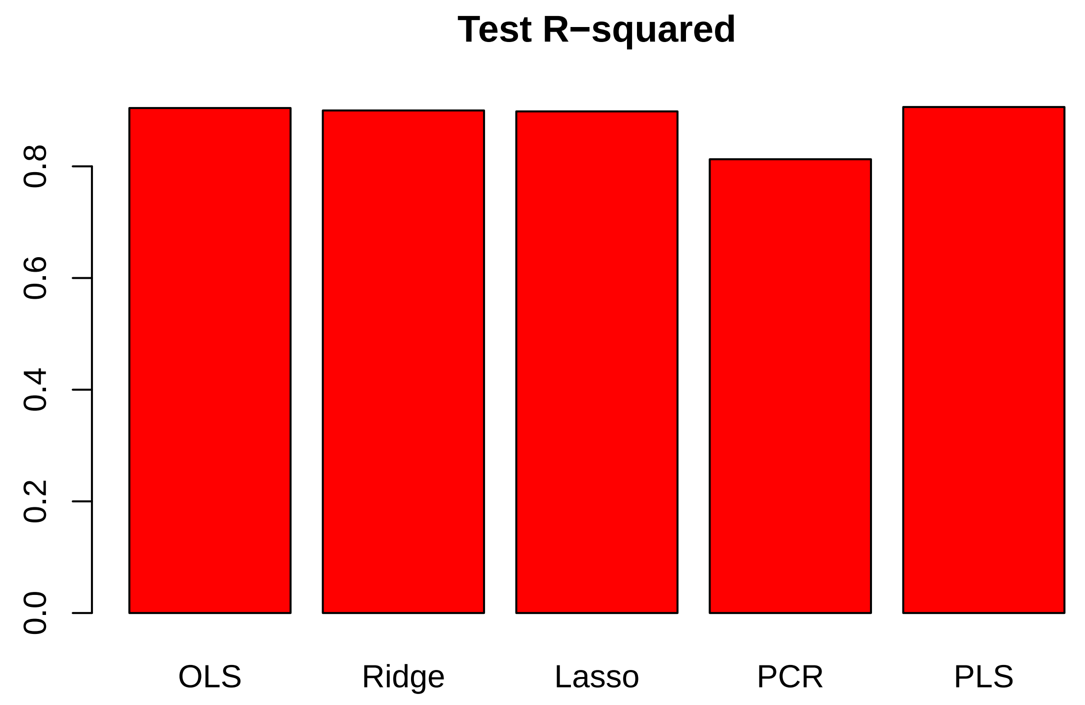
```
## [1] 1508987
```

Test RSS for PLS is about 15089871508987.

(g) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

Results for OLS, Lasso, Ridge are comparable. Lasso reduces the F.UndergradF.Undergrad and BooksBooks variables to zero and shrinks coefficients of other variables. Here are the test R2 for all models.

```
test.avg = mean(College.test[, "Apps"])
lm.test.r2 = 1 - mean((College.test[, "Apps"] - lm.pred)^2) /mean((College.test[, "Apps"] - test.avg)^2)
ridge.test.r2 = 1 - mean((College.test[, "Apps"] - ridge.pred)^2) /mean((College.test[, "Apps"] - test.a
lasso.test.r2 = 1 - mean((College.test[, "Apps"] - lasso.pred)^2) /mean((College.test[, "Apps"] - test.a
pcr.test.r2 = 1 - mean((College.test[, "Apps"] - data.frame(pcr.pred))^2) /mean((College.test[, "Apps"]
pls.test.r2 = 1 - mean((College.test[, "Apps"] - data.frame(pls.pred))^2) /mean((College.test[, "Apps"]
barplot(c(lm.test.r2, ridge.test.r2, lasso.test.r2, pcr.test.r2, pls.test.r2), col="red", names.arg=c("
```



The plot shows that test R2 for all models except PCR are around 0.9, with PLS having slightly higher test R2 than others. PCR has a smaller test R2 of less than 0.8. All models except PCR predict college applications with high accuracy.