Stat 154 - Quiz 3

Bret Hart

October 17, 2016

Question 1

First, we randomly sample a test set from the housing data to set aside and use as a means of predicting the test error, through the validation set approach.

This is done in the R script.

```
#set.seed(82)
#sample.set <- sample(506,455)
#housing.train <- housing[sample.set,]
#housing.test <- housing[-sample.set,]
```

Question 2

Next, we examine the full correlation matrix to carry out exploratory analysis and gain a better sense of each predictor's relationship with MEDV, our dependent variable.

Table 1: Correlation Matrix of MEDV on 12 Predictors

	X	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV
X	1.00	0.43	-0.10	0.41	-0.01	0.40	-0.10	0.21	-0.30	0.68	0.67	0.27	0.27	-0.23
CRIM	0.43	1.00	-0.21	0.42	-0.06	0.44	-0.27	0.37	-0.39	0.66	0.61	0.30	0.50	-0.41
ZN	-0.10	-0.21	1.00	-0.53	-0.06	-0.51	0.31	-0.57	0.67	-0.30	-0.31	-0.40	-0.42	0.36
INDUS	0.41	0.42	-0.53	1.00	0.07	0.76	-0.39	0.64	-0.69	0.59	0.72	0.38	0.61	-0.48
CHAS	-0.01	-0.06	-0.06	0.07	1.00	0.11	0.08	0.11	-0.11	-0.01	-0.03	-0.14	-0.03	0.15
NOX	0.40	0.44	-0.51	0.76	0.11	1.00	-0.29	0.73	-0.77	0.61	0.67	0.18	0.59	-0.42
RM	-0.10	-0.27	0.31	-0.39	0.08	-0.29	1.00	-0.24	0.20	-0.22	-0.30	-0.37	-0.61	0.70
AGE	0.21	0.37	-0.57	0.64	0.11	0.73	-0.24	1.00	-0.75	0.45	0.50	0.27	0.62	-0.38
DIS	-0.30	-0.39	0.67	-0.69	-0.11	-0.77	0.20	-0.75	1.00	-0.48	-0.52	-0.23	-0.50	0.24
RAD	0.68	0.66	-0.30	0.59	-0.01	0.61	-0.22	0.45	-0.48	1.00	0.91	0.45	0.49	-0.38
TAX	0.67	0.61	-0.31	0.72	-0.03	0.67	-0.30	0.50	-0.52	0.91	1.00	0.46	0.55	-0.46
PTRATIO	0.27	0.30	-0.40	0.38	-0.14	0.18	-0.37	0.27	-0.23	0.45	0.46	1.00	0.40	-0.52
LSTAT	0.27	0.50	-0.42	0.61	-0.03	0.59	-0.61	0.62	-0.50	0.49	0.55	0.40	1.00	-0.74
MEDV	-0.23	-0.41	0.36	-0.48	0.15	-0.42	0.70	-0.38	0.24	-0.38	-0.46	-0.52	-0.74	1.00

If we merely look at which predictors have the highest absolute R^2 value for the MEDV dependent variable, we can go down from abs(R^2) == 1 to 0.
We see LSTAT has -.74, which is the highest abs(R^2) value.
RM has the next highest abs(R^2), but is positive, at .70.

These are the only two predictors with abs(R^2) above .6. PTRATIO, with -.52 sneaks in as the last predictor with abs(R^2) above .5. These first 2 predictors, LSTAT and RM, and secondarily PTRATIO, seem as though they are the 3 most relevant predictors for predicting MEDV.

CRIM: -.41, INDUS: -.48, NOX: -.42, TAX: -.46 are the remaining predictors above .4 - while they may not be the greatest at predicting MEDV, they are still pretty good for a real-world data set. The rest of the predictors likely do not do a great job of predicting MEDV.

Question 3

Now we implement an algorithm to actually choose a certain combination of predictors to use in our model to predict MEDV.

We carry out a best subset selection. This entails testing every single combination of predictors for each number of possible predictors. While computationally intensive, with only 12 predictors, it's only testing around 4000 models, which takes around 5-10 seconds on my computer. My algorithm can be seen in my .R file, but here is the matrix of each best combination of predictors for each level, with attached R2 value. I used the R2 value to choose the best combination of predictors at every level.

```
print(interim.models)
```

```
##       [,1]     [,2]     [,3]      [,4]      [,5]      [,6]      [,7]
##  [1,] "LSTAT"  "RM"     "RM"      "RM"      "NOX"     "CHAS"    "ZN"
##  [2,] "0.5499" "LSTAT"  "PTRATIO" "DIS"     "RM"      "NOX"     "CHAS"
##  [3,] NA       "0.6458" "LSTAT"   "PTRATIO" "DIS"     "RM"      "NOX"
##  [4,] NA       NA       "0.684"   "LSTAT"   "PTRATIO" "DIS"     "RM"
##  [5,] NA       NA       NA        "0.697"   "LSTAT"   "PTRATIO" "DIS"
##  [6,] NA       NA       NA        NA        "0.7135"  "LSTAT"   "PTRATIO"
##  [7,] NA       NA       NA        NA        NA        "0.7191"  "LSTAT"
##  [8,] NA       NA       NA        NA        NA        NA        "0.7228"
##  [9,] NA       NA       NA        NA        NA        NA        NA
## [10,] NA       NA       NA        NA        NA        NA        NA
## [11,] NA       NA       NA        NA        NA        NA        NA
## [12,] NA       NA       NA        NA        NA        NA        NA
## [13,] NA       NA       NA        NA        NA        NA        NA
##       [,8]      [,9]      [,10]     [,11]     [,12]
##  [1,] "CRIM"    "CRIM"    "CRIM"    "CRIM"    "CRIM"
##  [2,] "ZN"      "ZN"      "ZN"      "ZN"      "ZN"
##  [3,] "CHAS"    "NOX"     "CHAS"    "CHAS"    "INDUS"
##  [4,] "NOX"     "RM"      "NOX"     "NOX"     "CHAS"
##  [5,] "RM"      "DIS"     "RM"      "RM"      "NOX"
##  [6,] "DIS"     "RAD"     "DIS"     "AGE"     "RM"
##  [7,] "PTRATIO" "TAX"     "RAD"     "DIS"     "AGE"
##  [8,] "LSTAT"   "PTRATIO" "TAX"     "RAD"     "DIS"
##  [9,] "0.7246"  "LSTAT"   "PTRATIO" "TAX"     "RAD"
## [10,] NA        "0.7286"  "LSTAT"   "PTRATIO" "TAX"
## [11,] NA        NA        "0.7332"  "LSTAT"   "PTRATIO"
## [12,] NA        NA        NA        "0.7335"  "LSTAT"
## [13,] NA        NA        NA        NA        "0.7335"
```

And here are the first few terms of x, the list which holds the lm object for each best combination of predictors at each level of BSS. Also, here are the first few terms of y, which holds the summary information for each lm object in x.

```
list(x[[1]],x[[2]],x[[3]])
```

```
## [[1]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
```

```
## Coefficients:
## (Intercept)        LSTAT
##     34.7761      -0.9728
##
##
## [[2]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)           RM        LSTAT
##     -1.0809       5.0851      -0.6596
##
##
## [[3]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)           RM      PTRATIO        LSTAT
##     18.8904       4.4903      -0.9291      -0.5841
```

```
list(y[[1]],y[[2]],y[[3]])
```

```
## [[1]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.211  -3.955  -1.366   1.924  24.494
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.77615    0.59375   58.57   <2e-16 ***
## LSTAT       -0.97278    0.04135  -23.52   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.151 on 453 degrees of freedom
## Multiple R-squared:  0.5499, Adjusted R-squared:  0.5489
## F-statistic: 553.4 on 1 and 453 DF,  p-value: < 2.2e-16
##
##
## [[2]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -18.1767  -3.4392  -0.9863   1.8928  27.9584
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.08086    3.28293  -0.329    0.742
## RM           5.08507    0.45953  11.066   <2e-16 ***
## LSTAT       -0.65961    0.04636 -14.228   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.463 on 452 degrees of freedom
## Multiple R-squared:  0.6458, Adjusted R-squared:  0.6443
## F-statistic: 412.1 on 2 and 452 DF,  p-value: < 2.2e-16
##
##
## [[3]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5579  -3.1383  -0.8254   1.7517  29.4643
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.89042    4.11803   4.587 5.83e-06 ***
## RM           4.49025    0.44195  10.160  < 2e-16 ***
## PTRATIO     -0.92907    0.12587  -7.381 7.62e-13 ***
## LSTAT       -0.58407    0.04502 -12.974  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.166 on 451 degrees of freedom
## Multiple R-squared:  0.684,  Adjusted R-squared:  0.6819
## F-statistic: 325.4 on 3 and 451 DF,  p-value: < 2.2e-16
```

Now that we have all best lm objects in x, summaries in y, and the matrix of each predictor combination/R2 value, we can compare the predictor lm combinations by using Cp, BIC, Adjusted R Squared, and Cross Validation to compare more accurately across predictor levels where R2 and RSE cannot.

First, here is our set of Cp statistics at each level.

```
cp.vect
```

```
##        [,1]
## 1  37.87574
## 2  29.94586
## 3  26.85178
## 4  25.86267
## 5  24.58304
## 6  24.21595
## 7  24.01179
## 8  23.95663
```

```
## 9  23.72542
## 10 23.43842
## 11 23.51850
## 12 23.61470
```

And we find the minimum Cp, which belongs to the 10 predictor lm object.

```
cp.min
```

```
##       10
## 23.43842
```

Now, check BIC, and find the minimum BIC across the lm objects. We find the BIC actually suggests the 6 predictor model, not the 10 predictor model.

```
bic.vect
```

```
##        [,1]
## 1  2960.775
## 2  2857.815
## 3  2812.045
## 4  2799.015
## 5  2779.665
## 6  2776.816
## 7  2776.993
## 8  2780.043
## 9  2779.559
## 10 2777.842
## 11 2783.538
## 12 2789.562
```

```
bic.min
```

```
##        6
## 2776.816
```

Now, Adjusted R2, and its minimum. We find that the adj r2 also suggests the 10 predictor model.

```
adj.rsq
```

```
##         [,1]
## 1  0.5488763
## 2  0.6442553
## 3  0.6818944
## 4  0.6943270
## 5  0.7103247
## 6  0.7153449
## 7  0.7184102
## 8  0.7196771
## 9  0.7230958
## 10 0.7272113
## 11 0.7268507
## 12 0.7262905
```

```
adj.max
```

```
##        10
## 0.7272113
```

Lastly, we check cross validation. The algorithm for this can be found in my source R code. Here we will just look at the matrix of outputs and their minimum. Again, we find that the 10 predictor model is probably the best.

```
cv.vect
```

```
##        [,1]
## 1  37.90992
## 2  30.65402
## 3  27.43080
## 4  26.54895
## 5  25.45592
## 6  25.17931
## 7  24.86339
## 8  24.94839
## 9  24.47136
## 10 24.23624
## 11 24.63550
## 12 24.67389
```

```
cv.min
```

```
##       10
## 24.23624
```

Despite the BIC implying otherwise, 3 of 4 statistics which adjust training error and simulate test error suggest that the 10 predictor model is best. Thus, we can pretty confidently assume that this is the best model under BSS. Let's look at the 10 predictor model.

```
x[[10]]
```

```
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)         CRIM           ZN         CHAS          NOX
##    41.36801     -0.11059      0.04350      2.51443    -18.09774
##          RM          DIS          RAD          TAX      PTRATIO
##     3.67410     -1.51076      0.25733     -0.01057     -0.93950
##       LSTAT
##    -0.57255
```

```
y[[10]]
```

```
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.1133  -2.8195  -0.4932   1.8069  26.0557
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  41.368014   5.156529   8.022 9.32e-15 ***
## CRIM         -0.110592   0.039231  -2.819 0.005033 **
## ZN            0.043504   0.014666   2.966 0.003177 **
## CHAS          2.514428   0.905335   2.777 0.005713 **
## NOX         -18.097739   3.789909  -4.775 2.44e-06 ***
## RM            3.674099   0.428019   8.584  < 2e-16 ***
## DIS          -1.510755   0.199139  -7.586 1.95e-13 ***
## RAD           0.257328   0.068009   3.784 0.000176 ***
## TAX          -0.010572   0.003578  -2.955 0.003296 **
## PTRATIO      -0.939496   0.140882  -6.669 7.69e-11 ***
## LSTAT        -0.572547   0.051505 -11.116  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.783 on 444 degrees of freedom
## Multiple R-squared:  0.7332, Adjusted R-squared:  0.7272
## F-statistic:   122 on 10 and 444 DF,  p-value: < 2.2e-16
```

Now that we have a good sense of what the 10 predictor model found by BSS looks and behaves like, and are relatively confident that it is the best of the 4000 or so models tested comprehensively, we can try other methods of regression and model selection to compare.

Question 4

Although we are doing forward selection next, it is entirely nested within BSS. Forward selection cannot find a 'better' model, it simply runs far less computations than BSS. However, we can compare the two to see if, in this case, forward selection will actually return a similar result at a fraction of the computing power!

We thus carry out forward selection. This entails finding the best predictor at a level, and then keeping that predictor in the model and subsequently testing every other predictor with it, ascertaining the best combination, and continuing from there. It's obviously far less complex, run-time wise, than BSS, but it has no guarantee of getting the same predictor set, as it is necessarily making compromises by keeping a predictor in the model at each level. The algorithm can be found in my .R file, but here is the matrix of each best added predictor model at each level, along with the respective R2 value. I used the R2 value to choose each best subsequent combination of predictors at every level.

```
print(interim.models2)
```

```
##       [,1]     [,2]     [,3]       [,4]       [,5]       [,6]       [,7]
##  [1,] "LSTAT"  "LSTAT"  "LSTAT"    "LSTAT"    "LSTAT"    "LSTAT"    "LSTAT"
##  [2,] "0.5499" "RM"     "RM"       "RM"       "RM"       "RM"       "RM"
##  [3,] NA       "0.6458" "PTRATIO"  "PTRATIO"  "PTRATIO"  "PTRATIO"  "PTRATIO"
##  [4,] NA       NA       "0.684"    "DIS"      "DIS"      "DIS"      "DIS"
```

```
##  [5,] NA       NA       NA       "0.697"   "NOX"     "NOX"     "NOX"
##  [6,] NA       NA       NA       NA        "0.7135"  "CHAS"    "CHAS"
##  [7,] NA       NA       NA       NA        NA        "0.7191"  "ZN"
##  [8,] NA       NA       NA       NA        NA        NA        "0.7228"
##  [9,] NA       NA       NA       NA        NA        NA        NA
## [10,] NA       NA       NA       NA        NA        NA        NA
## [11,] NA       NA       NA       NA        NA        NA        NA
## [12,] NA       NA       NA       NA        NA        NA        NA
## [13,] NA       NA       NA       NA        NA        NA        NA
##       [,8]      [,9]      [,10]     [,11]     [,12]
##  [1,] "LSTAT"   "LSTAT"   "LSTAT"   "LSTAT"   "LSTAT"
##  [2,] "RM"      "RM"      "RM"      "RM"      "RM"
##  [3,] "PTRATIO" "PTRATIO" "PTRATIO" "PTRATIO" "PTRATIO"
##  [4,] "DIS"     "DIS"     "DIS"     "DIS"     "DIS"
##  [5,] "NOX"     "NOX"     "NOX"     "NOX"     "NOX"
##  [6,] "CHAS"    "CHAS"    "CHAS"    "CHAS"    "CHAS"
##  [7,] "ZN"      "ZN"      "ZN"      "ZN"      "ZN"
##  [8,] "CRIM"    "CRIM"    "CRIM"    "CRIM"    "CRIM"
##  [9,] "0.7246"  "RAD"     "RAD"     "RAD"     "RAD"
## [10,] NA        "0.7286"  "TAX"     "TAX"     "TAX"
## [11,] NA        NA        "0.7332"  "AGE"     "AGE"
## [12,] NA        NA        NA        "0.7335"  "INDUS"
## [13,] NA        NA        NA        NA        "0.7335"
```

And here are the first few terms of xx, the list which holds the lm object for each best subsequently added combination of predictors at each level of forward selection. Also, here are the first few terms of yy, which holds the summary information for each lm object in xx.

```
list(xx[[1]],xx[[2]],xx[[3]])
```

```
## [[1]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)        LSTAT
##     34.7761      -0.9728
##
##
## [[2]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)        LSTAT           RM
##     -1.0809      -0.6596       5.0851
##
##
## [[3]]
##
## Call:
```

```
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)        LSTAT           RM      PTRATIO
##     18.8904      -0.5841       4.4903      -0.9291
```

```
list(yy[[1]],yy[[2]],yy[[3]])
```

```
## [[1]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.211  -3.955  -1.366   1.924  24.494
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.77615    0.59375   58.57   <2e-16 ***
## LSTAT       -0.97278    0.04135  -23.52   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.151 on 453 degrees of freedom
## Multiple R-squared:  0.5499, Adjusted R-squared:  0.5489
## F-statistic: 553.4 on 1 and 453 DF,  p-value: < 2.2e-16
##
##
## [[2]]
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.1767  -3.4392  -0.9863   1.8928  27.9584
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.08086    3.28293  -0.329    0.742
## RM           5.08507    0.45953  11.066   <2e-16 ***
## LSTAT       -0.65961    0.04636 -14.228   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.463 on 452 degrees of freedom
## Multiple R-squared:  0.6458, Adjusted R-squared:  0.6443
## F-statistic: 412.1 on 2 and 452 DF,  p-value: < 2.2e-16
##
##
## [[3]]
##
## Call:
```

```
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5579  -3.1383  -0.8254   1.7517  29.4643
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.89042    4.11803   4.587 5.83e-06 ***
## RM           4.49025    0.44195  10.160  < 2e-16 ***
## PTRATIO     -0.92907    0.12587  -7.381 7.62e-13 ***
## LSTAT       -0.58407    0.04502 -12.974  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.166 on 451 degrees of freedom
## Multiple R-squared:  0.684,  Adjusted R-squared:  0.6819
## F-statistic: 325.4 on 3 and 451 DF,  p-value: < 2.2e-16
```

At least for the first 3 lm objects in xx, they match exactly with the BSS lm objects!

Now that we have all best lm objects in xx, summaries in yy, and the matrix of each subsequent predictor combination/R2 value, we can compare the predictor lm combinations by using Cp, BIC, Adjusted R Squared, and Cross Validation to compare more accurately across predictor levels where R2 and RSE cannot.

First, here is our set of Cp statistics at each level.

```
cp.vect.for
```

```
##        [,1]
## 1  37.87574
## 2  29.94586
## 3  26.85178
## 4  25.86267
## 5  24.58304
## 6  24.21595
## 7  24.01179
## 8  23.95663
## 9  23.77656
## 10 23.43842
## 11 23.51850
## 12 23.61470
```

And we find the minimum Cp, which belongs to the 10 predictor lm object, just like the BSS Cp statistic! However, we haven't yet checked to see if the model has the same predictors. We'll do this after calculating the rest of the statistics.

```
cp.min.for
```

```
##       10
## 23.43842
```

Now, check BIC, and find the minimum BIC across the lm objects. We find the BIC actually suggests the 6 predictor model, just like in BSS. Again, we don't know if these models are exactly the same, but it's still

interesting that the BIC suggests the same model again. Perhaps this is due to its stricter penalty than the other statistics on number of predictors.

```
bic.vect.for
```

```
##        [,1]
## 1  2960.775
## 2  2857.815
## 3  2812.045
## 4  2799.015
## 5  2779.665
## 6  2776.816
## 7  2776.993
## 8  2780.043
## 9  2780.582
## 10 2777.842
## 11 2783.538
## 12 2789.562
```

```
bic.min.for
```

```
##        6
## 2776.816
```

Now, Adjusted R2, and its minimum. We find that the adj r2 also suggests the 10 predictor model, just as in BSS.

```
adj.rsq.for
```

```
##         [,1]
## 1  0.5488763
## 2  0.6442553
## 3  0.6818944
## 4  0.6943270
## 5  0.7103247
## 6  0.7153449
## 7  0.7184102
## 8  0.7196771
## 9  0.7230958
## 10 0.7272113
## 11 0.7268507
## 12 0.7262905
```

```
adj.max.for
```

```
##        10
## 0.7272113
```

Lastly, we check cross validation. The algorithm for this can be found in my source R code. Here we will just look at the matrix of outputs and their minimum. Again, just as before, we find that the 10 predictor model is probably the best.

```
cv.vect.for
```

```
##        [,1]
## 1  37.90992
## 2  30.65402
## 3  27.43080
## 4  26.54895
## 5  25.45592
## 6  25.17931
## 7  24.86339
## 8  24.94839
## 9  24.47136
## 10 24.23624
## 11 24.63550
## 12 24.67389
```

```
cv.min.for
```

```
##       10
## 24.23624
```

We arrive at the exact same results as in BSS! 3 of 4 statistics, with the BIC saying otherwise, suggest that the 10 predictor model is best, which gives us a pretty strong reason to believe that it is probably the best. Let's look at the two models and see if they're the same.

```
x[[10]]
```

```
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)         CRIM           ZN         CHAS          NOX
##    41.36801     -0.11059      0.04350      2.51443    -18.09774
##          RM          DIS          RAD          TAX      PTRATIO
##     3.67410     -1.51076      0.25733     -0.01057     -0.93950
##       LSTAT
##    -0.57255
```

```
xx[[10]]
```

```
##
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)        LSTAT           RM      PTRATIO          DIS
##    41.36801     -0.57255      3.67410     -0.93950     -1.51076
##         NOX         CHAS           ZN         CRIM          RAD
##   -18.09774      2.51443      0.04350     -0.11059      0.25733
##         TAX
##    -0.01057
```

Although it is a bit hard to parse, we can see that the models are actually equivalent in BSS and forward selection! And, while the predictors are the same, they were added in a different order!

Of course, the conclusion here is still the same. Forward selection gives us the same 10 predictor model as best subset selection, and if it is the literal best subset possible, it's pretty cool that we were able to get it with forward selection, a method of significantly less computational runtime!

Question 5

Both BSS and forward selection only examine models of pure multiple linear regression. Sure, they can examine a large number of multiple linear regression models and isolate the best ones, but they are testing and showcasing only one possible modeling type. We can try out other modes of regression, such as shrinkage methods like Ridge Regression and the Lasso to see what kinds of models can be produced by adjusting and shrinking coefficients in the multiple linear regression model including/testing all predictors.

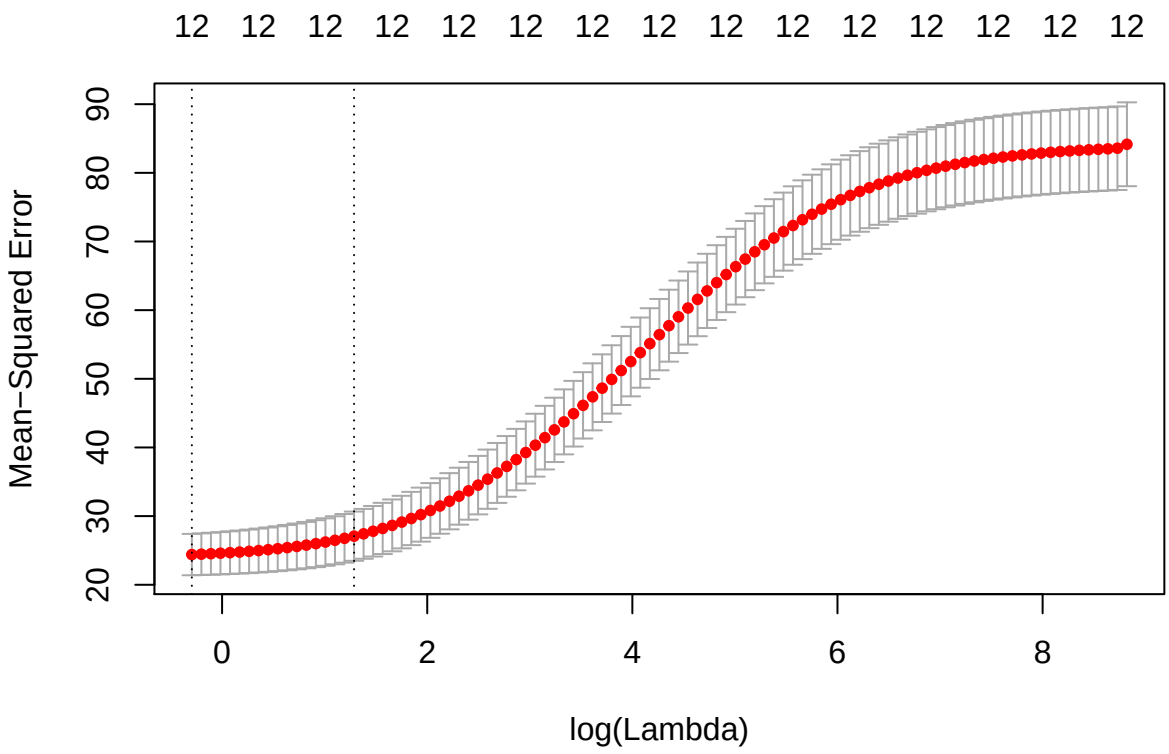
We use the package glmnet for this, and thus, do not have to create the algorithm to undergo this regression process. We can examine exactly the functions that create these regressions.

First, let's carry out a ridge regression. We will load these objects from our .R file, but we can still look at the code that was used to produce it. The alpha = 0 method tells glmnet to carry out a ridge regression. We have to use a model.matrix to map the x values, as glmnet is very particular with regards to what format the predictors take.

```
#ridge.reg <- glmnet(model.matrix(MEDV ~ ., housing.train2), MEDV, alpha = 0)
```

And, glmnet includes a function to carry out cv over all possible shrinkage penalties, and can allow us to examine the different lambdas and their associated MSE. Using this cv function, we can plot all of the lambdas tested and their MSEs, and even extract directly the 'best' lambda with best proportional MSE.

```
#ridge.cv <-  cv.glmnet(model.matrix(MEDV ~ ., housing.train2), MEDV, alpha = 0)
plot(ridge.cv)
```



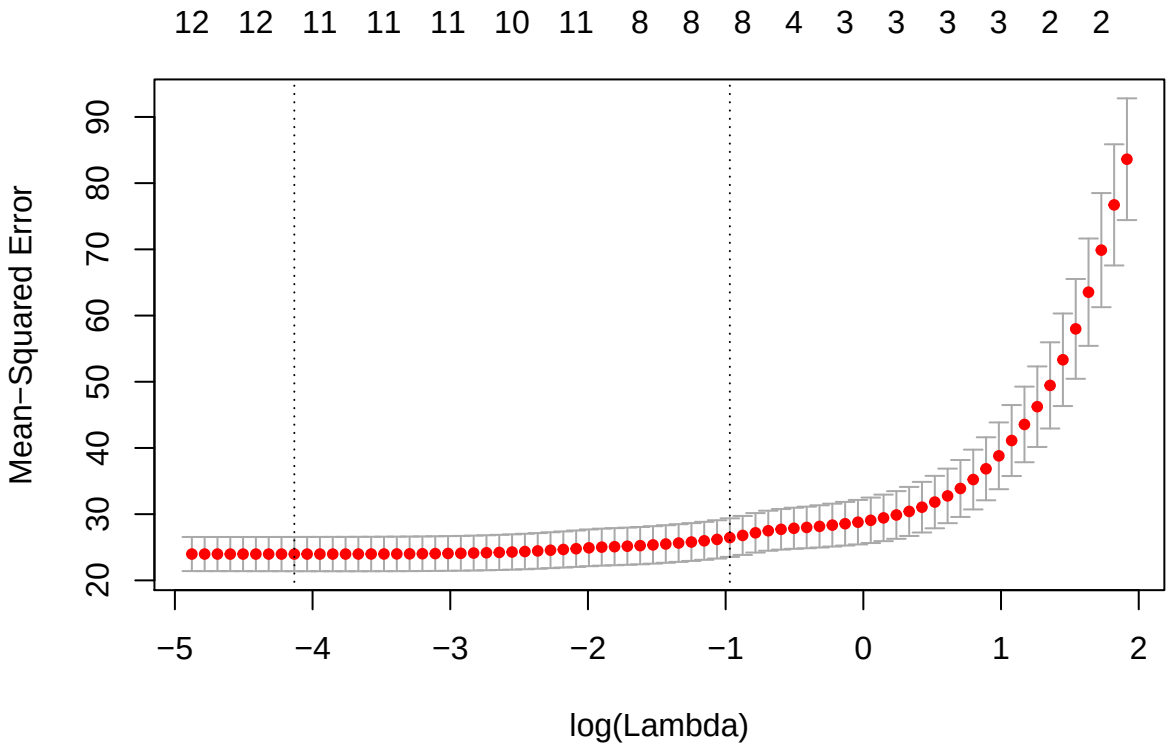

```
ridge.cv$lambda.min
```

```
## [1] 0.7445355
```

We find the minimizing and best lambda for the ridge regression is .7445355.

We can undergo a similar procedure for the lasso, with the only adjustment being simply changing the 'alpha =' method from 0 to 1. Thus, the only change is alpha = 1. Glmnet sure is useful! Again, we will extract the lasso objects from our .R source, but we can still look at the code as it is very simple.

```
#lasso.reg <- glmnet(model.matrix(MEDV ~ ., housing.train2), MEDV, alpha = 1)
#lasso.cv <-  cv.glmnet(model.matrix(MEDV ~ ., housing.train2), MEDV, alpha = 1)
plot(lasso.cv)
```




```
lasso.cv$lambda.min
```

```
## [1] 0.01604053
```

Although the cv lambda vs MSE plot is a bit hard to interpret in this case, we can still extract the best lambda shrinkage penalty directly from the lasso.cv object with the best proportional MSE. In this case, the minimizing and best lambda for the lasso regression is 0.01604053.

Thus, with these two shrinkage methods, we arrive at two additional regression models for the data which we could not have arrived at with simply multiple linear regression, as these methods actually adjust predictors to their calculated, proportional weights.

Question 6

Best subset and Forward selection both gave us the 10 predictor model as a favorite, with the BIC the sole believer in a 6 predictor model both times. Both the ridge and lasso regressions are quite different from this 10 predictor model, and we have little reason to think that the 6 predictor multiple linear regression model is very good. So, the best 3 we have are the 10 predictor and the ridge and lasso models. Let's try them and see how they do against the test data we set aside at the beginning!

First, let's look quickly at the 3 regression objects we ended up with.

```
x[[10]]
```

```
##
```

```
## Call:
## lm(formula = as.formula(paste("MEDV ~ ", abc, sep = "")))
##
## Coefficients:
## (Intercept)         CRIM           ZN         CHAS          NOX
##    41.36801     -0.11059      0.04350      2.51443    -18.09774
##          RM          DIS          RAD          TAX      PTRATIO
##     3.67410     -1.51076      0.25733     -0.01057     -0.93950
##       LSTAT
##    -0.57255
```

```
#x.matrix <- model.matrix(MEDV ~ ., housing.train2)
#ridge.lamb <- glmnet(x.matrix,MEDV, alpha = 0, lambda = ridge.cv$lambda.min)

ridge.lamb$beta
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                          s0
## (Intercept)   .
## CRIM         -0.089805841
## ZN            0.030694808
## INDUS        -0.040071592
## CHAS          2.592558675
## NOX         -12.602856722
## RM            3.877073427
## AGE           0.001170380
## DIS          -1.082721375
## RAD           0.123176072
## TAX          -0.005191358
## PTRATIO      -0.856896169
## LSTAT        -0.515839150
```

```
#lasso.lamb <- glmnet(x.matrix,MEDV, alpha = 1, lambda = lasso.cv$lambda.min)
lasso.lamb$beta
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                          s0
## (Intercept)   .
## CRIM         -0.103314585
## ZN            0.041783204
## INDUS         .
## CHAS          2.455643457
## NOX         -17.948597448
## RM            3.667483722
## AGE           0.006671939
## DIS          -1.426115538
## RAD           0.233548370
## TAX          -0.009527020
## PTRATIO      -0.936861951
## LSTAT        -0.582266427
```

These aren't the greatest ways of viewing the differing regression objects, but we can examine how good they are better by testing their predictive capability against the set aside testing data. Let's look at the ridge and lasso regressions first.

With the ridge regression, we do a little bit of input manipulation, then end up with the a vector of the predicted values from the observations in the test data, from the ridge regression. We use the glmnet function predict() to do this. Thus, we end up with a vector of predicted values for MEDV, for the test data, derived and predicted from our ridge regression with minimized lambda.

```
#test.matrix <- model.matrix(housing.test$MEDV ~., housing.test)
#ridge.pred <- predict(ridge.reg, s = ridge.cv$lambda.min, newx = test.matrix[,-2])

head(ridge.pred)
```

```
##           1
## 5  28.49820
## 9  12.05049
## 10 19.55905
## 12 21.95347
## 34 15.05794
## 67 24.98682
```

It should be the same length as the test MEDV vector, and it is! Thus, we subsetted our data correctly and can use the test data set aside to simulate a test error!

```
length(ridge.pred)==length(housing.test$MEDV)
```

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

Here are the first few residuals (i.e, the predicted values from the ridge regression - the actual MEDV values from the test data). And, from this, we can find the MSE of the ridge regression prediction, that is, the error in prediction from predicted vs actual values.

```
head(ridge.pred - housing.test$MEDV)
```

```
##             1
## 5  -7.7018035
## 9  -4.4495125
## 10  0.6590452
## 12  3.0534715
## 34  1.9579373
## 67  5.5868177
```

```
mean((ridge.pred - housing.test$MEDV)^2)
```

```
## [1] 24.45394
```

Thus, our MSE for the ridge regression prediction is 24.45394. We want the lowest one possible. Let's see how the other models do and compare their MSEs.

The lasso regression follows very similarly from the ridge regression, with the only difference in derivation being the alpha and lambda values in predict(). We create a vector of predicted values from the test observations from the lasso regression, examine the residuals of predicted - actual, and calculate the MSE of the lasso regression.

```
#lasso.pred <- predict(lasso.reg, s = lasso.cv$lambda.min, newx = test.matrix[,-2])
head(lasso.pred)
```

```
##           1
## 5  28.09458
## 9  10.70758
## 10 18.73264
## 12 21.48585
## 34 14.43739
## 67 25.28400
```

```
head(lasso.pred - housing.test$MEDV)
```

```
##             1
## 5  -8.1054221
## 9  -5.7924235
## 10 -0.1673644
## 12  2.5858478
## 34  1.3373941
## 67  5.8840005
```

```
mean((lasso.pred - housing.test$MEDV)^2)
```

```
## [1] 24.17735
```

Our MSE for the lasso regression prediction is 24.17735. This is very marginally lower than the ridge regression, so perhaps the lasso is doing a slightly better job of predicting and modeling the data - but, of course, this subtle of a difference could be due entirely to the specific test data which we used. Before we make any conclusions, let's look at the 10 predictor model from the BSS.

The predict() function for a straightforward multiple linear regression is a little simpler, so we can observe it directly. We'll still load it from our .R source, but the inputs are simply the lm object and the data it's testing/predicting on.

```
#bss.pred <- predict(x[[10]], newdata = housing.test)
```

Just as with the lasso and ridge predictions, we can look at some of the predicted values, look at some of the residuals, and calculate the MSE, the error of prediction from predicted vs. actual.

```
length(bss.pred) == length(housing.test$MEDV)
```

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

```
head(bss.pred - housing.test$MEDV)
```

```
##          5          9         10         12         34         67
## -8.2234141 -6.0124912 -0.4621988  2.3068718  1.1765322  5.9919176
```

```
mean((bss.pred - housing.test$MEDV)^2)
```

```
## [1] 23.77013
```

We end up with the lowest MSE, of 23.77013!

For kicks, let's try out the 6 predictor model that the BIC wants us to try to see if we're really missing something crucial.

```
#bss.pred6 <- predict(x[[6]], newdata = housing.test)
length(bss.pred6) == length(housing.test$MEDV)
```

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

```
head(bss.pred6 - housing.test$MEDV)
```

```
##         5         9        10        12        34        67
## -7.084572 -5.997339  0.173475  2.895990  1.231207  4.212888
```

```
mean((bss.pred6 - housing.test$MEDV)^2)
```

```
## [1] 28.63179
```

As we suspected, however, this 6 predictor model isn't nearly as good as our other models. The MSE of 28.63179 is a good deal worse than the other models. Sorry, BIC!

Some thoughts/conclusions/the paper

Thus, it seems that when it comes to this specific test data set, straightforward multiple linear regression gives us the best model, that of a specific combination of 10 of the 12 predictors. Of course, we could've tested an infinite number of lambdas for our ridge and lasso regressions to get a slightly better result, but this could perhaps overfit or simply go on for an indeterminate amount of time. It's hard to say why BSS did the best in this case, as it's always complicated to assess the role of so many predictors and such complex math behind each of the regression formats, but it's rewarding as a programmer because the BSS was much harder to implement and program!

It's important to keep in mind that these 3 models were all fitted from a specific subset of our training data and tested on a set of arbitrary test data from our training data. Who knows what the predictive capability or MSE of the different regressions would be when put against different test data sets, or other real-world data points. We have created through this process 3 very similiar in quality regression models, which, perhaps could be viewed simultaneously to predict and assess possible future data points or methods of action. We don't necessarily have to use just 1! Also, realistically, we would use a package for BSS and forward selection to make these algorithms a bit more feasible on the programming end, making it easier to test and examine many different kinds of models and consider their respective pros and cons.

Some interesting things to note are the large coefficient of NOX in all 3 models despite its predicted low coefficient; the fact that the lasso drops the INDUS variable; each predictor in the 10 predictor model has a significant t-value; the 2 variables dropped from the 10 predictor model, (AGE: relative age of houses and INDUS: how residential a neighborhood is) seem to play an intuitive role in determining housing prices, yet, they apparently are not significant enough to truly predict prices; and the relative similarity of each model's MSE despite their vastly different coefficient estimates for each predictor. Data is complicated and there is no

such thing as a "best" model. It's important to consider many different ones and view them in juxtaposition and conjunction with each other to ascertain the best sense of the data possible.

When viewing my findings vs. the findings of the paper, a few things come to mind: they had an additional variable, INC, which seems to have very high correlation with MEDV - higher correlation than any of the predictors which we used. This possibly could make their data set better suited to predicting, as they had fewer confounding (i.e., in multiple linear regression, missing/unnacounted for) variables. They don't talk about it too much explicitly, but it seems like it informed a good deal of their research. The formula they used is a seemingly manually derived adjustment to the least squares regression line upon certain predictors, with weights that I believe they derived through some complex calculations, but are each implemented manually. They didn't just include the coefficients directly: they used the log of DIS, RAD, and STAT, squared RM, and included a few predictors I think they created themselves, as some sort of combination of the given predictors. Their R2 of the finalized model is .82, vs the R2 of our 10-predictor model, .7332. :(This is a significanlty better R2! But, they definitely put a huge amount of work into this project, so I hope they could do something a bit better than just the most basic model selection algorithms. Naturally, this is an intimidating regression model to compare my models to!

They also discuss at decent length the role they believe each predictor plays: some offer very interesting insight into what makes a house worth what it sells for on the market. Some intuitive predictors are CRIM - crime rate, INDUS - how much industry vs. residential living there is in a neighborhood, and NOX - how toxic the air is. All of these seem to have a reasonable role in determining housing costs. There are some variables which make sense, but seem to practically be strokes of genius to include in this model, in my mind, these are very creative inclusions: CHAS - whether the area is near the local river or not, DIS - a weighted combination of distances to 5 employment centers, RAD - how easy it is to get to nearby highways, and PTRATIO - pupil vs teacher ratio. These seem to have an intuitive role in determining housing prices, but thinking of them, calculating them, and including them in this model really lends some creativity and genius to this prediction. This, to me, is inspiring of the role statisticians can play in the real world - I'm sure they worked closely with community and public policy experts, politicians, etc. to gain a sense of what variables may be less obvious but play a profound role in predicting housing costs. Of course, I find it hard to believe that any of these will really play a more significant role than the LSTAT - the percent of citizens of lower status in the population, but not including them in the model would be irresponsible and short-sighted. We, as statisticians, cannot forget that there can be creativity in all steps of the statistical modeling process, and that we can branch out, interdisciplinarily, to gain a better intution towards what predictors there may be or what our model can really achieve. There is a lot to think about here.