Stat 495 - Regularization Lab - Ridge and LASSO - Solution

A.S. Wagaman

Ridge Regression

This first example is adapted from ISLR, pages 251-255, from the reproduction of that lab by McNamara and Crouser.

```
data(Hitters)
# ?Hitters # MLB data from 1986 and 1987; 322 obs on 20 variables
```

We want to predict Salary, which is missing for 59 players. The ridge regression function won't automatically remove NAs, so we do it.

```
Hitters <- na.omit(Hitters) #263 obs now
```

We demo the new functions and code below. Ideally, we'd split into a training/test data set here, but we'll do that below after checking out these functions.

The function we'll be using is *glmnet* which can do more than just ridge regression, or *cv.glmnet* for cross-validation.

The format these functions require is not the usual Y ~ X format we are used to. Instead, we need to pass in the X matrix, and Y vector.

```
x <- model.matrix(Salary ~ ., Hitters)[, -1] # trim off the first column

y <- Hitters %>% select(Salary) %>%  unlist() %>%  as.numeric()
```

The `model.matrix()` function is particularly useful for creating X; not only does it produce a matrix corresponding to the 19 predictors but it also automatically transforms any qualitative variables into dummy variables as shown below (if you aren't sure what this means, please ask). The latter property is important because `glmnet()` can only take numerical, quantitative inputs.

```
head(Hitters)
```

```
##                    AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Alan Ashby          315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis         479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson        496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga    321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin     594  169     4   74  51    35    11   4408  1133     19
## -Al Newman           185   37     1   23   8    21     2    214    42      1
##                    CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Alan Ashby          321  414    375      N        W     632      43     10
## -Alvin Davis         224  266    263      A        W     880      82     14
## -Andre Dawson        828  838    354      N        E     200      11      3
## -Andres Galarraga     48   46     33      N        E     805      40      4
## -Alfredo Griffin     501  336    194      A        W     282     421     25
## -Al Newman            30    9     24      N        E      76     127      7
##                    Salary NewLeague
```

```
## -Alan Ashby        475.0          N
## -Alvin Davis       480.0          A
## -Andre Dawson      500.0          N
## -Andres Galarraga   91.5          N
## -Alfredo Griffin   750.0          A
## -Al Newman          70.0          A
```

```r
head(x)
```

```
##                    AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Alan Ashby          315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis         479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson        496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga    321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin     594  169     4   74  51    35    11   4408  1133     19
## -Al Newman           185   37     1   23   8    21     2    214    42      1
##                    CRuns CRBI CWalks LeagueN DivisionW PutOuts Assists Errors
## -Alan Ashby          321  414    375       1         1     632      43     10
## -Alvin Davis         224  266    263       0         1     880      82     14
## -Andre Dawson        828  838    354       1         0     200      11      3
## -Andres Galarraga     48   46     33       1         0     805      40      4
## -Alfredo Griffin     501  336    194       0         1     282     421     25
## -Al Newman            30    9     24       1         0      76     127      7
##                    NewLeagueN
## -Alan Ashby                 1
## -Alvin Davis                0
## -Andre Dawson               1
## -Andres Galarraga           1
## -Alfredo Griffin            0
## -Al Newman                  0
```

Fitting the Ridge Model via an Example

glmnet can be used to fit several types of models, including both ridge and LASSO, so there is an argument to the function governing what type of model is fit. That parameter is α, and you want $\alpha = 0$ for ridge regression, so you will see that below.

Most of the text and code for the rest of this example is taken from the McNamara and Crouser reproduced lab, though it has been edited to change assignment operators to <- instead of =, and other similar changes/extra comments.

```r
grid <- 10^seq(10, -2, length = 100)
ridge_mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

By default the `glmnet()` function performs ridge regression for an automatically selected range of λ values. However, here we have chosen to implement the function over a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit.

(Note: the function has a pretty good automatic range of λ values, so you can also just let it use the default. This is designed to show you how you could specify these values if you wanted.)

As we will see, we can also compute model fits for a particular value of λ that is not one of the original grid values. Note that by default, the `glmnet()` function standardizes the variables so that they are on the same scale. To turn off this default setting, use the argument `standardize = FALSE`.

(Note: this can have severe implications for your coefficient estimates, which is why TRUE is the default.)

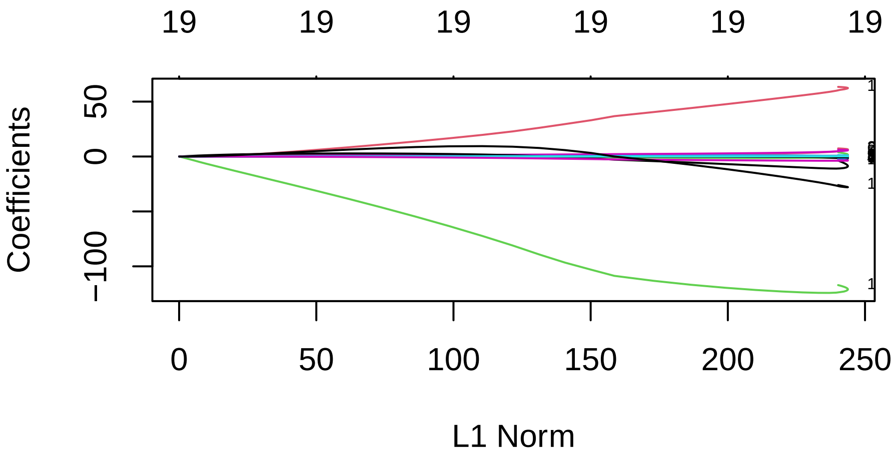
Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()`. In this case, it is a 20×100 matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ).

We can look at the path of the solution with a plot. Note that the plot has the l_1 norm (for the sum of the betas) on the x-axis, not the value of λ. $\lambda = 0$ is the OLS estimates on the far right, while $\lambda = \infty$ is the l_1 norm of 0 on the far left.

```
dim(coef(ridge_mod))
```

```
## [1]  20 100
```

```
plot(ridge_mod, label = TRUE)    # Draw plot of coefficients; can label them by variable #
```



We expect the coefficient estimates to be much smaller, in terms of l_2 norm, when a large value of λ is used, as compared to when a small value of λ is used. These are the coefficients when $\lambda = 11497.57$, along with their l_2 norm:

```
ridge_mod$lambda[50] #Display 50th lambda value
```

```
## [1] 11497.57
```

```
coef(ridge_mod)[, 50] # Display coefficients associated with 50th lambda value
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## 407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
##           RBI         Walks         Years        CAtBat         CHits
##   0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##   0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
##  -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```
sqrt(sum(coef(ridge_mod)[-1, 50]^2)) # Calculate l2 norm
```

```
## [1] 6.360612
```

In contrast, here are the coefficients when $\lambda = 705.4802$, along with their l_2 norm. Note the much larger l_2 norm of the coefficients associated with this smaller value of λ.

```
ridge_mod$lambda[60] #Display 60th lambda value
```

```
## [1] 705.4802
```

```r
coef(ridge_mod)[, 60] # Display coefficients associated with 60th lambda value
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs           RBI
##   54.32519950    0.11211115    0.65622409    1.17980910    0.93769713    0.84718546
##         Walks         Years        CAtBat         CHits        CHmRun         CRuns
##    1.31987948    2.59640425    0.01083413    0.04674557    0.33777318    0.09355528
##          CRBI        CWalks       LeagueN     DivisionW       PutOuts       Assists
##    0.09780402    0.07189612   13.68370191  -54.65877750    0.11852289    0.01606037
##        Errors    NewLeagueN
##   -0.70358655    8.61181213
```

```r
sqrt(sum(coef(ridge_mod)[-1, 60]^2)) # Calculate l2 norm
```

```
## [1] 57.11001
```

We can use the `predict()` function for a number of purposes. For instance, we can obtain the ridge regression coefficients for a new value of λ, say 50:

```r
round(predict(ridge_mod, s = 50, type = "coefficients")[1:20, ], 3)
```

```
## (Intercept)       AtBat        Hits       HmRun        Runs         RBI
##      48.766      -0.358       1.969      -1.278       1.146       0.804
##       Walks       Years      CAtBat       CHits      CHmRun       CRuns
##       2.716      -6.218       0.005       0.106       0.624       0.221
##        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists
##       0.219      -0.150      45.926    -118.201       0.250       0.122
##      Errors  NewLeagueN
##      -3.279      -9.497
```

We now split the samples into a training set and a test set in order to estimate the test error of ridge regression. This uses a 50-50 split, although splits such as 80-20 or 70-30 are more common. There are multiple ways to do this.

```r
set.seed(1) #set in ISLR lab
n <- nrow(Hitters)
train_index <- sample(1:n, 0.5 * n)
test_index <- setdiff(1:n, train_index)

train <- Hitters[train_index, ]
test <- Hitters[test_index, ]

#further set up for using glmnet
x_train <- model.matrix(Salary ~ ., train)[, -1]
x_test <- model.matrix(Salary ~ ., test)[, -1]

y_train <- train %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()

y_test <- test %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()
```

Getting the data into the format for glmnet is most of the code here. If you were just running *lm* or *glm*, you would be fine using the train and test data sets constructed.

Next we fit a ridge regression model on the training set, and evaluate its MSE on the test set, using $\lambda = 4$, as an example. There is no reason to think this is a "good" lambda - it's just for illustration. Note the use of the `predict()` function again: this time we get predictions for a test set, by replacing `type="coefficients"` with the `newx` argument.

(Note: Be careful with arguments, for some predict functions, you need *newx* and others are *newdata*, etc. Use the help files to assist you.)

```r
# fit model on training set
ridge_mod <- glmnet(x_train, y_train, alpha = 0, lambda = grid, thresh = 1e-12)
#get predictions on test set using model
ridge_pred <- predict(ridge_mod, s = 4, newx = x_test)
# compute MSE on test set
mean((ridge_pred - y_test)^2)
```

```
## [1] 142199.2
```

The test MSE is 142199.2. Note that if we had instead simply fit a model with just an intercept, we would have predicted each test observation using the mean of the training observations. In that case, we could compute the test set MSE like this:

```r
mean((mean(y_train) - y_test)^2)
```

```
## [1] 224669.9
```

Comparing these MSEs tells us that the ridge regression is doing better than a model with no predictors.

(If you have any questions about this comparison, please ask for assistance.)

To compare MSEs, we could also get MSE for a model with just an intercept by fitting a ridge regression model with a very large value of λ. Note that `1e10` means 10^{10}.

```r
# we already fit the set of models
# just get predictions with our chosen lambda
ridge_pred <- predict(ridge_mod, s = 1e10, newx = x_test)
# then compute MSE
mean((ridge_pred - y_test)^2)
```

```
## [1] 224669.8
```

So fitting a ridge regression model with $\lambda = 4$ leads to a much lower test MSE than fitting a model with just an intercept. We now check whether there is any benefit to performing ridge regression with $\lambda = 4$ instead of just performing least squares regression. Recall that least squares is simply ridge regression with $\lambda = 0$.

Note: In order for `glmnet()` to yield the exact least squares coefficients when $\lambda = 0$, we use the argument `exact=T` when calling the `predict()` function. Otherwise, the `predict()` function will interpolate over the grid of λ values used in fitting the `glmnet()` model, yielding approximate results. Even when we use `exact = T`, there remains a slight discrepancy in the third decimal place between the output of `glmnet()` when $\lambda = 0$ and the output of `lm()`; this is due to numerical approximation on the part of `glmnet()`.

```r
#compare coefficients first
lm(Salary ~ ., data = train)
```

```
##
## Call:
## lm(formula = Salary ~ ., data = train)
##
## Coefficients:
## (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##    274.0145      -0.3521      -1.6377       5.8145       1.5424       1.1243
##       Walks        Years       CAtBat        CHits       CHmRun        CRuns
```

```
##      3.7287     -16.3773      -0.6412       3.1632       3.4008      -0.9739
##        CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##     -0.6005       0.3379     119.1486    -144.0831       0.1976       0.6804
##      Errors   NewLeagueN
##     -4.7128     -71.0951
```

```
predict(ridge_mod, s = 0, exact = T,  x = x_train, y = y_train, type = "coefficients")[1:20, ]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  274.0200994   -0.3521900   -1.6371383    5.8146692    1.5423361    1.1241837
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##    3.7288406  -16.3795195   -0.6411235    3.1629444    3.4005281   -0.9739405
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   -0.6003976    0.3378422  119.1434637 -144.0853061    0.1976300    0.6804200
##       Errors   NewLeagueN
##   -4.7127879  -71.0898914
```

```
#this code required some editing due to changes in the functions. The training data had to be added as
ridge_pred <- predict(ridge_mod, s = 0, newx = x_test, exact = T, x = x_train, y = y_train)
mean((ridge_pred - y_test)^2)
```

```
## [1] 168588.6
```

It looks like we are indeed improving over regular least-squares!

What two values are being compared to argue this?

The test set MSE for the lm and for the ridge regression are being compared. The values are 168588.6 for the lm and 142199.2 for the ridge regression with $\lambda = 4$.

Side note: in general, if we want to fit a (unpenalized) least squares model, then we should use the `lm()` function, since that function provides more useful outputs, such as standard errors and p-values for the coefficients.

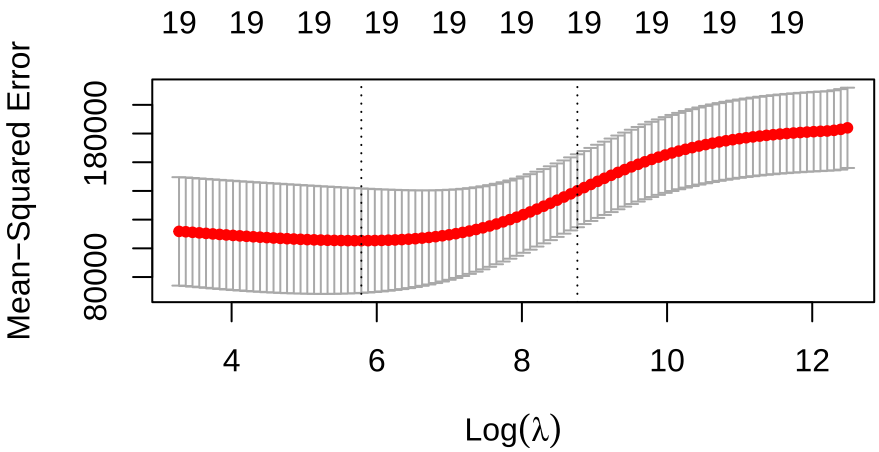
Instead of arbitrarily choosing $\lambda = 4$, it would be better to use cross-validation to choose the tuning parameter λ. We can do this using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs 10-fold cross-validation, though this can be changed using the argument `folds`. Note that we set a random seed first so our results will be reproducible, since the choice of the cross-validation folds is random.

```
set.seed(1)
cv.out <- cv.glmnet(x_train, y_train, alpha = 0) # Fit ridge regression model on training data
bestlam <- cv.out$lambda.min  # Select lambda that minimizes training MSE
bestlam
```

```
## [1] 326.0828
```

Therefore, we see that the value of λ that results in the smallest cross-validation error is 326.0828. We can also plot the MSE as a function of $\log(\lambda)$:

```
plot(cv.out) # Draw plot of training MSE as a function of log(lambda)
```



Now, we want to find out what the test MSE associated with this value of λ is.

```
ridge_pred <- predict(ridge_mod, s = bestlam, newx = x_test) # Use best lambda to predict test data
mean((ridge_pred - y_test)^2) # Calculate test MSE
```

```
## [1] 139856.6
```

This represents a further improvement over the test MSE that we got using $\lambda = 4$. Finally, we examine the coefficient estimates for our ridge regression on the training data set, using the best lambda we selected via cross-validation. You can do this by running predict on our cv.glmnet object or re-fitting the model using glmnet.

```
# uses cv.glmnet object
predict(cv.out, type = "coefficients", s = bestlam)[1:20,]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  66.96586695   0.03504444   0.31510871   2.62854376   0.67807546   0.72960727
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.94298810  -1.72698509   0.01303118   0.06723721   0.67875720   0.12990766
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.16222494   0.17496469  35.01522536 -87.62017045   0.09991737   0.08226860
##       Errors   NewLeagueN
##  -0.47684379  18.57525550
```

```
#re-fits on training first
out <- glmnet(x_train, y_train, alpha = 0) # Fit ridge regression model on training dataset
predict(out, type = "coefficients", s = bestlam)[1:20,] # Display coefficients using lambda chosen by C
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  66.96586695   0.03504444   0.31510871   2.62854376   0.67807546   0.72960727
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.94298810  -1.72698509   0.01303118   0.06723721   0.67875720   0.12990766
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.16222494   0.17496469  35.01522536 -87.62017045   0.09991737   0.08226860
##       Errors   NewLeagueN
##  -0.47684379  18.57525550
```

As expected, none of the coefficients are exactly zero - ridge regression does not perform variable selection!

Important: The presentation of the functions and order of analysis here does not match what you would typically do in practice. It was designed to introduce you to the functions and how they work. What do you think the normal order of operations is? For example, would you spend a lot of time looking at coefficients for lambda values you have chosen at random?

Here is a better sequence of analysis tasks:

- Determine what analysis you want to perform and how the data needs to be set up for that.
- Create a train/test split if you are doing predictive modeling and want to assess performance using the test set.
- Fit the model on the training set, using appropriate functions to set values for tuning parameters.
- Assess performance on the test set (as appropriate).

Example Lasso Fitting

Lasso stands for least absolute shrinkage and selection operator and was developed by Tibshirani.

We can fit these models with either the lars package or glmnet. In glmnet, the only change from the ridge code above is that you have to set the alpha to 1 to run LASSO.

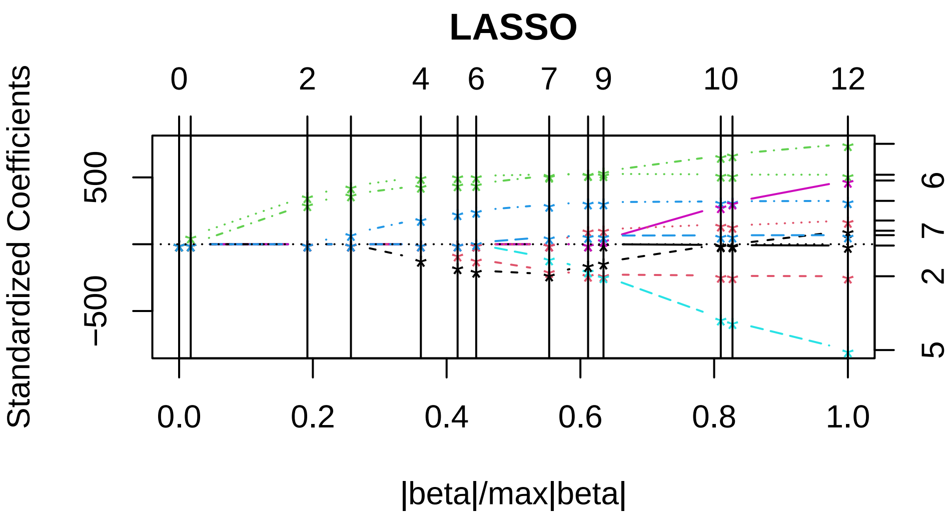
```
data(diabetes)
head(diabetes, 2)
```

```
##         x.age        x.sex        x.bmi        x.map         x.tc        x.ldl
## 1  0.038075906  0.050680119  0.061696207  0.021872355 -0.044223498 -0.034820763
## 2 -0.001882017 -0.044641637 -0.051474061 -0.026327835 -0.008448724 -0.019163340
##         x.hdl        x.tch        x.ltg        x.glu   y        x2.age
## 1 -0.043400846 -0.002592262  0.019908421 -0.017646125 151  0.0380759064
## 2  0.074411564 -0.039493383 -0.068329744 -0.092204050  75 -0.0018820165
##         x2.sex        x2.bmi        x2.map         x2.tc        x2.ldl
## 1  0.0506801187  0.0616962065  0.0218723550 -0.0442234984 -0.0348207628
## 2 -0.0446416365 -0.0514740612 -0.0263278347 -0.0084487241 -0.0191633397
##         x2.hdl        x2.tch        x2.ltg        x2.glu      x2.age^2
## 1 -0.0434008457 -0.0025922620  0.0199084209 -0.0176461252 -0.0148551625
## 2  0.0744115641 -0.0394933829 -0.0683297436 -0.0922040496 -0.0412915429
##       x2.bmi^2      x2.map^2       x2.tc^2      x2.ldl^2      x2.hdl^2
## 1  0.0225045739 -0.0310446765 -0.0043311197 -0.0137399243 -0.0046314248
## 2  0.0056427733 -0.0273076609 -0.0309389016 -0.0248010319  0.0400365241
##       x2.tch^2      x2.ltg^2      x2.glu^2    x2.age:sex    x2.age:bmi
## 1 -0.0304484629 -0.0288162192 -0.0275255618  0.0328649758  0.0405716741
## 2 -0.0094854824  0.0371612444  0.0880219609 -0.0066099928 -0.0067648038
##     x2.age:map     x2.age:tc    x2.age:ldl    x2.age:hdl    x2.age:tch
## 1  0.0016606410 -0.0465532511 -0.0382447104 -0.0345115069 -0.0121122609
## 2 -0.0159342243 -0.0117287997 -0.0096555406  0.0006995477 -0.0083689963
##     x2.age:ltg    x2.age:glu    x2.sex:bmi    x2.sex:map     x2.sex:tc
## 1  0.0030648916 -0.0302775066  0.0621030123  0.0122820371 -0.0485722318
## 2 -0.0102016795 -0.0113801440  0.0445181909  0.0137392710  0.0062226212
##     x2.sex:ldl    x2.sex:hdl    x2.sex:tch    x2.sex:ltg    x2.sex:glu
## 1 -0.0441240238 -0.0308042981 -0.0195479919  0.0142268228 -0.0293859648
## 2  0.0112617659 -0.0565675488  0.0224021267  0.0575880019  0.0784642525
##     x2.bmi:map     x2.bmi:tc    x2.bmi:ldl    x2.bmi:hdl    x2.bmi:tch
## 1  0.0090008988 -0.0717180778 -0.0603176794 -0.0413575386 -0.0240342004
## 2  0.0091148702 -0.0028355367  0.0087097314 -0.0671553344  0.0240456899
##     x2.bmi:ltg    x2.bmi:glu     x2.map:tc    x2.map:ldl    x2.map:hdl
## 1  0.0048261936 -0.0410278367 -0.0327209536 -0.0257474212 -0.0112074291
## 2  0.0552994440  0.0806093115 -0.0070399803  0.0018462404 -0.0319794277
##     x2.map:tch    x2.map:ltg    x2.map:glu     x2.tc:ldl     x2.tc:hdl
## 1 -0.0138251131 -0.0101938361 -0.0257784633 -0.0068689647  0.0398230899
## 2  0.0098745200  0.0203696547  0.0313619644 -0.0262353123 -0.0164622948
##      x2.tc:tch     x2.tc:ltg     x2.tc:glu    x2.ldl:hdl    x2.ldl:tch
## 1 -0.0193030537 -0.0428915072  0.0009629700  0.0423549304 -0.0220378305
```

```
## 2 -0.0155012002 -0.0123430995  0.0009326831 -0.0212564243 -0.0115642516
##      x2.ldl:ltg    x2.ldl:glu    x2.hdl:tch    x2.hdl:ltg    x2.hdl:glu
## 1 -0.0311245646 -0.0009221095  0.0334936252  0.0008521487  0.0311502576
## 2  0.0129733755  0.0237834359 -0.0238146613 -0.0945055990 -0.1403775894
##      x2.tch:ltg    x2.tch:glu    x2.ltg:glu
## 1 -0.0281911757 -0.0176581553 -0.0277936831
## 2  0.0252977155  0.0530335390  0.1040132768
```

We demo the lasso code on the same data set used in the original LARS paper (2002) with the lars package. Note the setup (you can check the help file for details) uses a matrix of predictors and then the response as we saw before with glmnet.

```r
object1 <- with(diabetes, lars(x, y, type = "lasso"))
plot(object1)
```




```r
coef(object1)
```

```
##               age        sex       bmi       map        tc       ldl       hdl
##   [1,]   0.000000    0.00000   0.00000   0.00000    0.0000   0.00000    0.0000
##   [2,]   0.000000    0.00000  60.11927   0.00000    0.0000   0.00000    0.0000
##   [3,]   0.000000    0.00000 361.89461   0.00000    0.0000   0.00000    0.0000
##   [4,]   0.000000    0.00000 434.75796  79.23645    0.0000   0.00000    0.0000
##   [5,]   0.000000    0.00000 505.65956 191.26988    0.0000   0.00000 -114.1010
##   [6,]   0.000000  -74.91651 511.34807 234.15462    0.0000   0.00000 -169.7114
##   [7,]   0.000000 -111.97855 512.04409 252.52702    0.0000   0.00000 -196.0454
##   [8,]   0.000000 -197.75650 522.26485 297.15974 -103.9462   0.00000 -223.9260
##   [9,]   0.000000 -226.13366 526.88547 314.38927 -195.1058   0.00000 -152.4773
##  [10,]   0.000000 -227.17580 526.39059 314.95047 -237.3410  33.62827 -134.5994
##  [11,]  -5.718948 -234.39762 522.64879 320.34255 -554.2663 286.73617    0.0000
##  [12,]  -7.011245 -237.10079 521.07513 321.54903 -580.4386 313.86213    0.0000
##  [13,] -10.012198 -239.81909 519.83979 324.39043 -792.1842 476.74584  101.0446
##             tch      ltg      glu
##   [1,]   0.0000   0.0000  0.00000
##   [2,]   0.0000   0.0000  0.00000
##   [3,]   0.0000 301.7753  0.00000
##   [4,]   0.0000 374.9158  0.00000
##   [5,]   0.0000 439.6649  0.00000
##   [6,]   0.0000 450.6674  0.00000
##   [7,]   0.0000 452.3927 12.07815
```

```
##  [8,]   0.0000 514.7495 54.76768
##  [9,] 106.3428 529.9160 64.48742
## [10,] 111.3841 545.4826 64.60667
## [11,] 148.9004 663.0333 66.33096
## [12,] 139.8579 674.9366 67.17940
## [13,] 177.0642 751.2793 67.62539
```

```r
summary(object1)
```

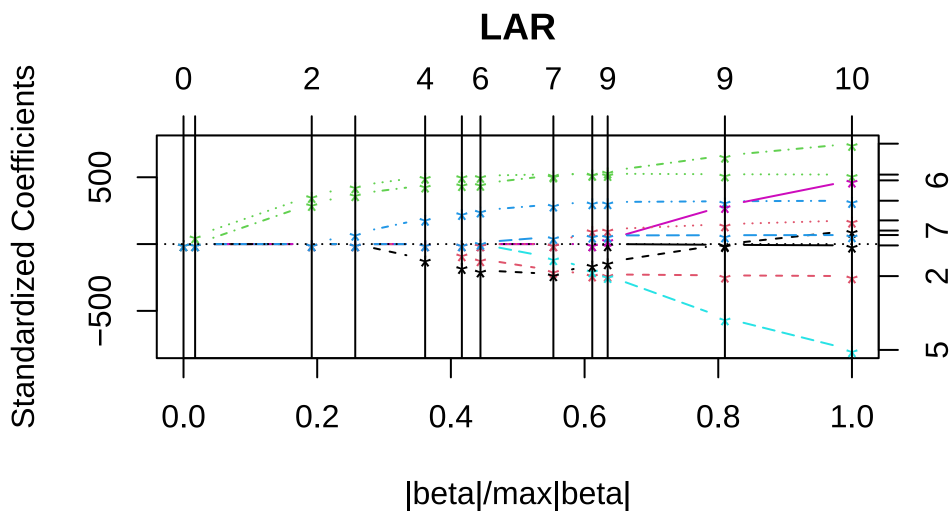
```
## LARS/LASSO
## Call: lars(x = x, y = y, type = "lasso")
##    Df     Rss       Cp
## 0   1 2621009 453.7263
## 1   2 2510465 418.0322
## 2   3 1700369 143.8012
## 3   4 1527165  86.7411
## 4   5 1365734  33.6957
## 5   6 1324118  21.5052
## 6   7 1308932  18.3270
## 7   8 1275355   8.8775
## 8   9 1270233   9.1311
## 9  10 1269390  10.8435
## 10 11 1264977  11.3390
## 11 10 1264765   9.2668
## 12 11 1263983  11.0000
```

The type you set has implications for the solution. The lasso option is the default. But you can also get the lar (without the lasso modification for if a nonzero coefficient hits zero) solution and forward stagewise solution (forward but in very small steps) via the lars algorithm (not quite forward selection).

```r
object2 <- with(diabetes, lars(x, y, type = "lar"))
summary(object2)
```

```
## LARS/LAR
## Call: lars(x = x, y = y, type = "lar")
##    Df     Rss       Cp
## 0   1 2621009 453.7263
## 1   2 2510465 418.0322
## 2   3 1700369 143.8012
## 3   4 1527165  86.7411
## 4   5 1365734  33.6957
## 5   6 1324118  21.5052
## 6   7 1308932  18.3270
## 7   8 1275355   8.8775
## 8   9 1270233   9.1311
## 9  10 1269390  10.8435
## 10 11 1263983  11.0000
```

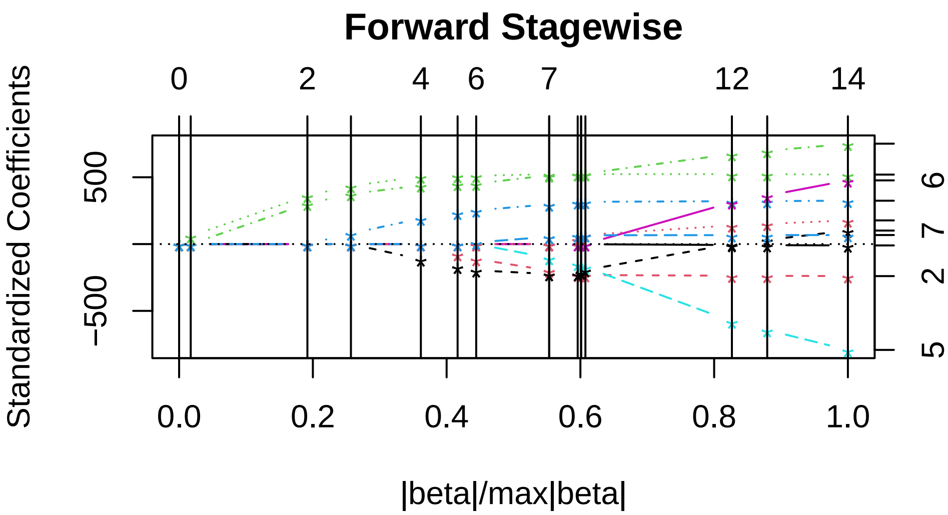
```r
plot(object2)
```




```r
object3 <- with(diabetes, lars(x, y, type = "forward.stagewise"))
summary(object3)
```

```
## LARS/Forward Stagewise
## Call: lars(x = x, y = y, type = "forward.stagewise")
##    Df     Rss       Cp
## 0   1 2621009 453.7263
## 1   2 2510465 418.0322
## 2   3 1700369 143.8012
## 3   4 1527165  86.7411
## 4   5 1365734  33.6957
## 5   6 1324118  21.5052
## 6   7 1308932  18.3270
## 7   8 1275355   8.8775
## 8   7 1275355   6.8775
## 9   7 1271599   5.5970
## 10  8 1271154   7.4452
## 11  9 1271150   9.4437
## 12 10 1270685  11.2853
## 13 10 1264371   9.1322
## 14 11 1263983  11.0000
```

```r
plot(object3)
```



What differences do you see in the solutions here based on type? Which models would you pick for each option of "type"? How do those models differ?

Your turn - Boston crime

Now that you've seen how to fit ridge and lasso models, let's try it out on another data set. Work in groups of 2 or 3 to help each other and discuss your results.

Your goal is to predict per capita crime rate in the Boston data set using ridge regression, lasso, and MLR model(s) of your choosing. How do the models compare? Which do you prefer? Feel free to look up the help file on the data set for more information about the variables.

```
Boston <- MASS::Boston #do NOT load the MASS library - it causes conflicts with dplyr
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

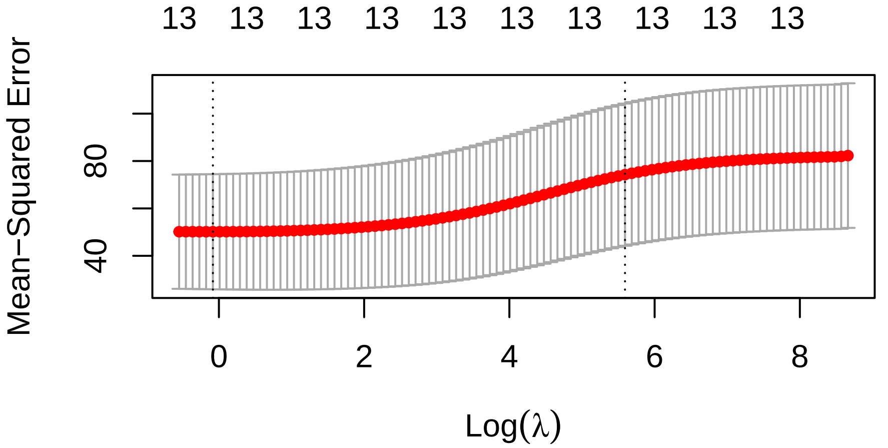
a) Start by fitting and comparing ridge and LASSO regressions. The Ridge regression (chapter 7 for code) has been fit for you, with a training data set of 50% of the observations.

```
set.seed(495)
xBos <- model.matrix(crim ~ ., Boston)[ , -1]
yBos <- Boston$crim
grid <- 10^seq(10, -2, length = 100)

n <- nrow(Boston)
train_index <- sample(1:n, 0.5 * n)
test_index <- setdiff(1:n, train_index)
trainBos <- Boston[train_index, ]
testBos <- Boston[test_index, ]
yBos.test <- yBos[test_index]
```

That was the data setup for the glmnet function. Here is the ridge fit with cross-validation used to choose the tuning parameter.

```
ridgeBos.mod <- glmnet(xBos[train_index,], yBos[train_index], alpha = 0, lambda = grid)
set.seed(495)
cvBos.out <- cv.glmnet(xBos[train_index,], yBos[train_index], alpha = 0)
plot(cvBos.out)
```




```
bestlamBos <- cvBos.out$lambda.min
bestlamBos
```

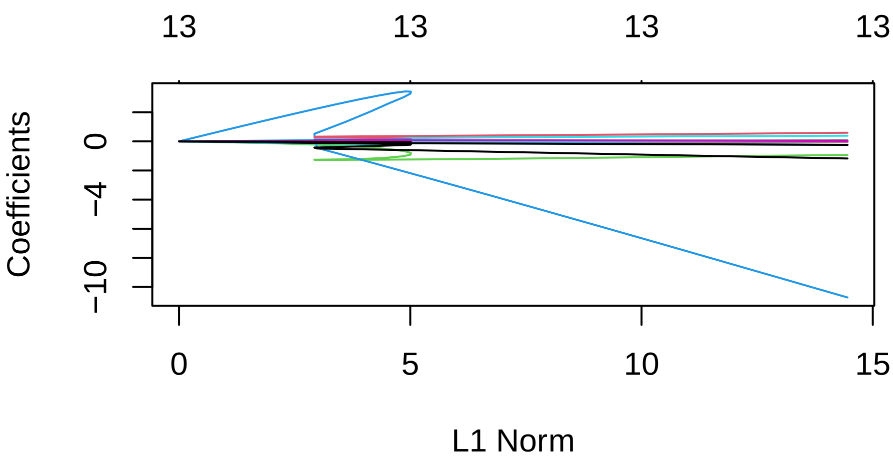
```
## [1] 0.9204819
```

Once we have the best lambda chosen by CV, we can look at the test MSE, etc.

```
ridgeBos.pred <- predict(ridgeBos.mod, s = bestlamBos, newx = xBos[test_index,])
mean((ridgeBos.pred - yBos.test)^2)
```

```
## [1] 37.06965
```

```
plot(ridgeBos.mod)
```



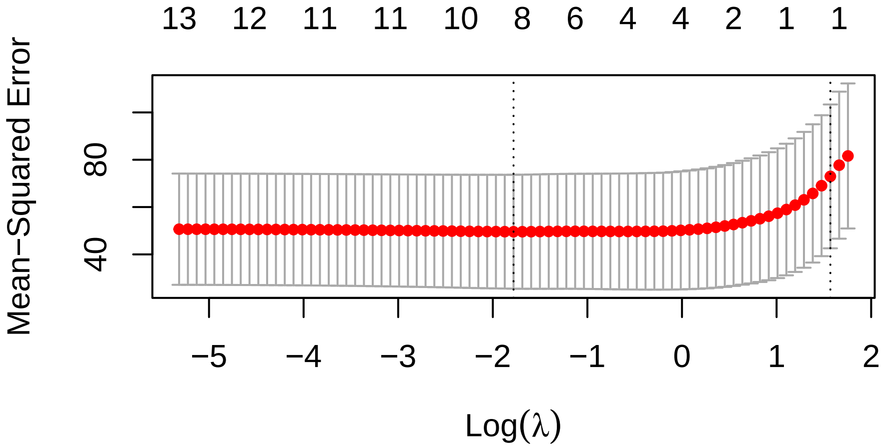

```
#coef(ridgeBos.mod) #can see coefs for all lambdas...
predict(ridgeBos.mod, type = "coefficients", s = bestlamBos)[1:14,]
```

```
##   (Intercept)            zn         indus          chas           nox 
##  5.5716002575  0.0333394098 -0.0713980874 -1.2060875906 -3.5101599562 
##            rm           age           dis           rad           tax 
##  0.2692659199  0.0005949055 -0.6935933659  0.4013525645  0.0065558070 
##       ptratio         black         lstat          medv 
## -0.0844368959 -0.0034696313  0.0780256120 -0.1454922702
```

Now you should fit the lasso model. Try fitting it via both the glmnet (change the alpha!) and lars functions.

```
# Example using glmnet
lassoBos.mod <- glmnet(xBos[train_index,], yBos[train_index], alpha = 1, lambda = grid) #make sure alph
set.seed(495)
```

```r
cvlBos.out <- cv.glmnet(xBos[train_index,], yBos[train_index], alpha = 1)
plot(cvlBos.out)
```




```r
bestlamBosl <- cvlBos.out$lambda.min
bestlamBosl
```

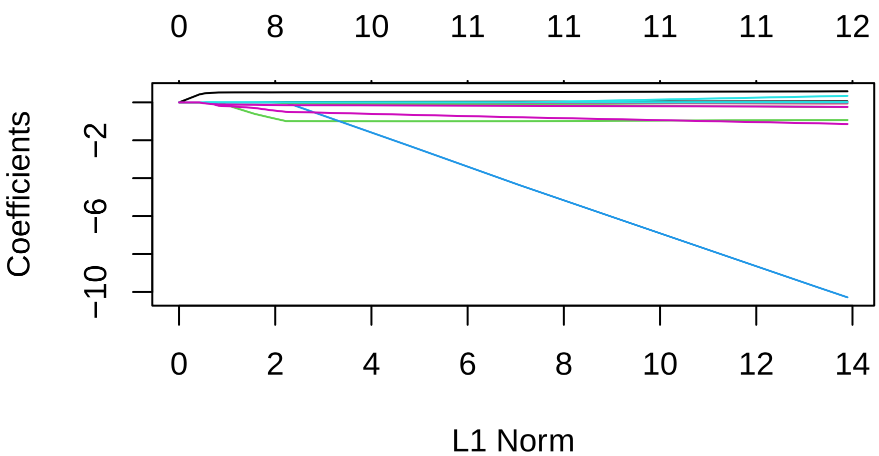
```
## [1] 0.1685161
```

```r
lassoBos.pred <- predict(lassoBos.mod, s = bestlamBosl, newx = xBos[test_index,])
mean((lassoBos.pred - yBos.test)^2)
```

```
## [1] 38.13944
```

```r
plot(lassoBos.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```




```r
#coef(lassoBos.mod) #can see coefs for all lambdas...
predict(lassoBos.mod, type = "coefficients", s = bestlamBosl)[1:14,]
```

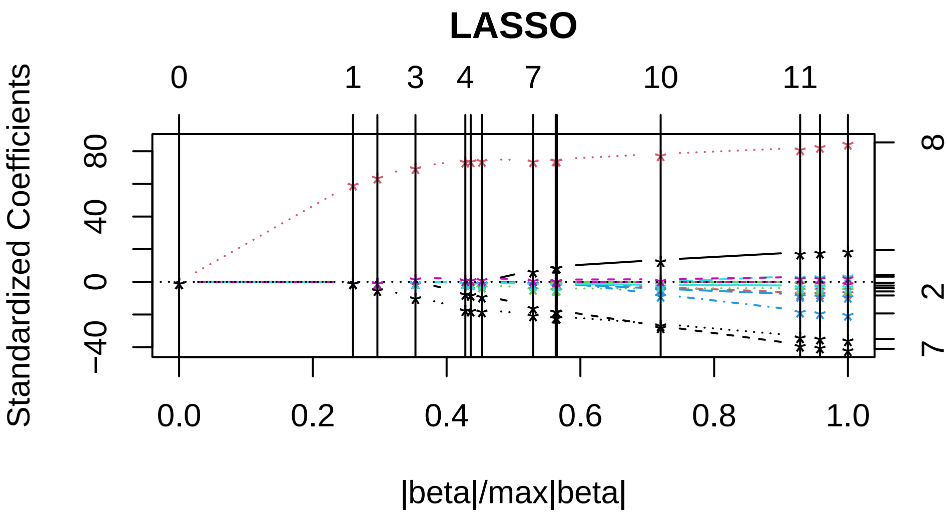
```
##   (Intercept)            zn         indus          chas           nox
##  4.0243228173  0.0266213396  0.0000000000 -0.9628083324  0.0000000000
##            rm           age           dis           rad           tax
##  0.0000000000  0.0000000000 -0.4846309202  0.5296616452  0.0000000000
##       ptratio         black         lstat          medv
## -0.0277282406 -0.0007388868  0.0130952796 -0.1430847333
```

What differences do you see in the models? Between the ridge and lasso fits?

Interestingly, the lasso sets several coefs to 0 - indus, nox, rm, age, and tax, with the magnitudes of others are very close to 0. Most coef signs agree between the two except for those now set to 0. The rad and ptratio coefficients are larger in magnitude in the lasso solution than the ridge solution. The biggest difference is the number of predictors removed by setting coefs to 0.

Now we fit this with the lars function.

```r
object1 <- lars(xBos[train_index,], yBos[train_index], type = "lasso")
plot(object1)
```




```r
coef(object1)
```

```
##                zn         indus       chas        nox        rm          age
##   [1,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##   [2,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##   [3,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##   [4,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##   [5,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##   [6,] 0.00000000  0.0000000000 -0.1705839   0.000000 0.0000000  0.000000000
##   [7,] 0.00000000  0.0000000000 -0.5041943   0.000000 0.0000000  0.000000000
##   [8,] 0.02146919  0.0000000000 -0.8511458   0.000000 0.0000000  0.000000000
##   [9,] 0.02834320  0.0000000000 -0.9988239   0.000000 0.0000000  0.000000000
##  [10,] 0.02852912 -0.0008257349 -1.0007274   0.000000 0.0000000  0.000000000
##  [11,] 0.04040833 -0.0295401576 -0.9896618  -4.348283 0.0000000  0.000000000
##  [12,] 0.05430297 -0.0593992311 -0.9262653  -9.839344 0.3201078  0.000000000
##  [13,] 0.05613465 -0.0568281566 -0.9299485 -10.334743 0.3480618  0.000000000
##  [14,] 0.05832425 -0.0527022437 -0.9284335 -10.923722 0.3918343 -0.001290323
##               dis       rad          tax     ptratio         black      lstat
##   [1,]  0.0000000 0.0000000  0.0000000000  0.00000000  0.0000000000 0.00000000
##   [2,]  0.0000000 0.4265529  0.0000000000  0.00000000  0.0000000000 0.00000000
##   [3,]  0.0000000 0.4564340  0.0000000000  0.00000000  0.0000000000 0.00000000
##   [4,]  0.0000000 0.4977271  0.0000000000  0.00000000  0.0000000000 0.02183975
##   [5,] -0.1915697 0.5256421  0.0000000000  0.00000000  0.0000000000 0.01604890
##   [6,] -0.2071892 0.5275220  0.0000000000  0.00000000  0.0000000000 0.01679081
##   [7,] -0.2365056 0.5296309  0.0000000000  0.00000000 -0.0003942663 0.01847918
##   [8,] -0.4357190 0.5274604  0.0000000000  0.00000000 -0.0007204631 0.01520349
##   [9,] -0.5009725 0.5304677  0.0000000000 -0.03593176 -0.0007444358 0.01210251
##  [10,] -0.5040861 0.5306390  0.0000000000 -0.03653268 -0.0007475213 0.01207628
```

```
## [11,] -0.7820336 0.5535160  0.0000000000 -0.12529176 -0.0011816146 0.01398595
## [12,] -1.1079211 0.5790340  0.0000000000 -0.23214381 -0.0014405932 0.02573914
## [13,] -1.1397975 0.5894863 -0.0006048272 -0.24047545 -0.0014565335 0.02597571
## [14,] -1.1859710 0.6030802 -0.0014124019 -0.25118738 -0.0014579609 0.02794936
##              medv
##  [1,]  0.00000000
##  [2,]  0.00000000
##  [3,] -0.02918987
##  [4,] -0.06298399
##  [5,] -0.11400403
##  [6,] -0.11592466
##  [7,] -0.11882644
##  [8,] -0.13585254
##  [9,] -0.14555330
## [10,] -0.14592230
## [11,] -0.17744138
## [12,] -0.22818815
## [13,] -0.23368471
## [14,] -0.24100168
```

```
summary(object1)
```

```
## LARS/LASSO
## Call: lars(x = xBos[train_index, ], y = yBos[train_index], type = "lasso")
##    Df   Rss       Cp
## 0   1 20701 187.3016
## 1   2 13240  31.3322
## 2   3 12756  23.0778
## 3   4 12195  13.1962
## 4   5 11783   6.4756
## 5   6 11758   7.9462
## 6   7 11713   9.0031
## 7   8 11567   7.9062
## 8   9 11522   8.9574
## 9  10 11520  10.9184
## 10 11 11377   9.8827
## 11 12 11292  10.0909
## 12 13 11290  12.0307
## 13 14 11288  14.0000
```

It looks like the 9 labeled model (row 8, df = 9) is similar to what we got with the best lambda approach with the glmnet function. However, based on Cp values, we might choose the 5th labeled model (df= 5, lowest Cp) which has even more coefs set to 0.

```
summary(object1)[9,]
```

```
## LARS/LASSO
## Call: lars(x = xBos[train_index, ], y = yBos[train_index], type = "lasso")
##   Df   Rss     Cp
## 8  9 11522 8.9574
```

```
coef(object1)[9,]
```

```
##            zn         indus          chas           nox            rm
##  0.0283431995  0.0000000000 -0.9988238630  0.0000000000  0.0000000000
##           age           dis           rad           tax       ptratio
##  0.0000000000 -0.5009724568  0.5304677174  0.0000000000 -0.0359317602
```

```
##         black         lstat          medv
## -0.0007444358  0.0121025094 -0.1455533013
```

```r
summary(object1)[5,]
```

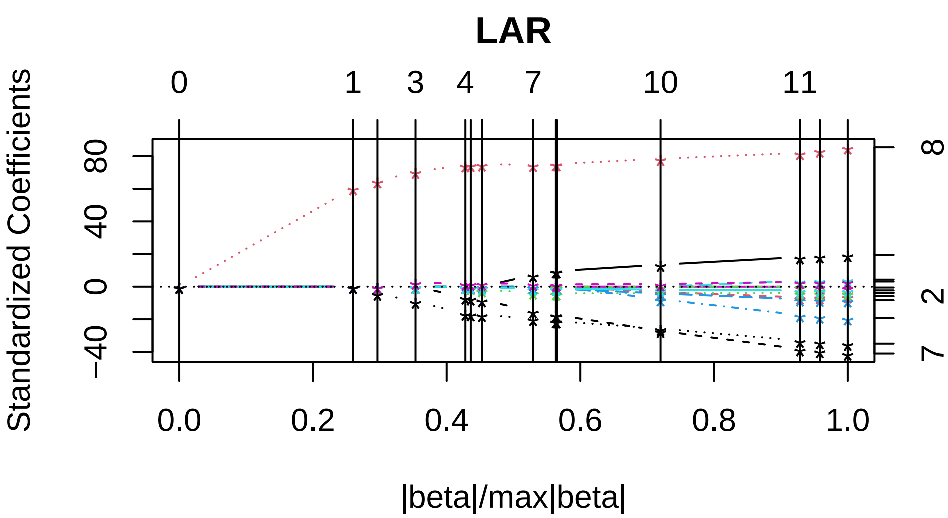
```
## LARS/LASSO
## Call: lars(x = xBos[train_index, ], y = yBos[train_index], type = "lasso")
##   Df   Rss     Cp
## 4  5 11783 6.4756
```

```r
coef(object1)[5,]
```

```
##         zn      indus       chas        nox         rm        age        dis
##  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000 -0.1915697
##        rad        tax    ptratio      black      lstat       medv
##  0.5256421  0.0000000  0.0000000  0.0000000  0.0160489 -0.1140040
```

We could change the type argument and see the lar solution to see if that differs:

```r
object2 <- lars(xBos[train_index,], yBos[train_index], type = "lar")
plot(object2)
```




```r
coef(object2)
```

```
##                zn         indus       chas        nox        rm          age
##  [1,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##  [2,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##  [3,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##  [4,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##  [5,] 0.00000000  0.0000000000  0.0000000   0.000000 0.0000000  0.000000000
##  [6,] 0.00000000  0.0000000000 -0.1705839   0.000000 0.0000000  0.000000000
##  [7,] 0.00000000  0.0000000000 -0.5041943   0.000000 0.0000000  0.000000000
##  [8,] 0.02146919  0.0000000000 -0.8511458   0.000000 0.0000000  0.000000000
##  [9,] 0.02834320  0.0000000000 -0.9988239   0.000000 0.0000000  0.000000000
## [10,] 0.02852912 -0.0008257349 -1.0007274   0.000000 0.0000000  0.000000000
## [11,] 0.04040833 -0.0295401576 -0.9896618  -4.348283 0.0000000  0.000000000
## [12,] 0.05430297 -0.0593992311 -0.9262653  -9.839344 0.3201078  0.000000000
## [13,] 0.05613465 -0.0568281566 -0.9299485 -10.334743 0.3480618  0.000000000
## [14,] 0.05832425 -0.0527022437 -0.9284335 -10.923722 0.3918343 -0.001290323
##               dis       rad           tax    ptratio          black      lstat
##  [1,]  0.0000000 0.0000000  0.0000000000  0.00000000  0.0000000000 0.00000000
```

```
##  [2,]  0.0000000 0.4265529  0.0000000000  0.00000000  0.0000000000 0.00000000
##  [3,]  0.0000000 0.4564340  0.0000000000  0.00000000  0.0000000000 0.00000000
##  [4,]  0.0000000 0.4977271  0.0000000000  0.00000000  0.0000000000 0.02183975
##  [5,] -0.1915697 0.5256421  0.0000000000  0.00000000  0.0000000000 0.01604890
##  [6,] -0.2071892 0.5275220  0.0000000000  0.00000000  0.0000000000 0.01679081
##  [7,] -0.2365056 0.5296309  0.0000000000  0.00000000 -0.0003942663 0.01847918
##  [8,] -0.4357190 0.5274604  0.0000000000  0.00000000 -0.0007204631 0.01520349
##  [9,] -0.5009725 0.5304677  0.0000000000 -0.03593176 -0.0007444358 0.01210251
## [10,] -0.5040861 0.5306390  0.0000000000 -0.03653268 -0.0007475213 0.01207628
## [11,] -0.7820336 0.5535160  0.0000000000 -0.12529176 -0.0011816146 0.01398595
## [12,] -1.1079211 0.5790340  0.0000000000 -0.23214381 -0.0014405932 0.02573914
## [13,] -1.1397975 0.5894863 -0.0006048272 -0.24047545 -0.0014565335 0.02597571
## [14,] -1.1859710 0.6030802 -0.0014124019 -0.25118738 -0.0014579609 0.02794936
##              medv
##  [1,]  0.00000000
##  [2,]  0.00000000
##  [3,] -0.02918987
##  [4,] -0.06298399
##  [5,] -0.11400403
##  [6,] -0.11592466
##  [7,] -0.11882644
##  [8,] -0.13585254
##  [9,] -0.14555330
## [10,] -0.14592230
## [11,] -0.17744138
## [12,] -0.22818815
## [13,] -0.23368471
## [14,] -0.24100168
```

```r
summary(object2)
```

```
## LARS/LAR
## Call: lars(x = xBos[train_index, ], y = yBos[train_index], type = "lar")
##    Df   Rss       Cp
## 0   1 20701 187.3016
## 1   2 13240  31.3322
## 2   3 12756  23.0778
## 3   4 12195  13.1962
## 4   5 11783   6.4756
## 5   6 11758   7.9462
## 6   7 11713   9.0031
## 7   8 11567   7.9062
## 8   9 11522   8.9574
## 9  10 11520  10.9184
## 10 11 11377   9.8827
## 11 12 11292  10.0909
## 12 13 11290  12.0307
## 13 14 11288  14.0000
```

Doesn’t appear to differ here.

```r
summary(object2)[5,]
```

```
## LARS/LAR
## Call: lars(x = xBos[train_index, ], y = yBos[train_index], type = "lar")
##   Df   Rss     Cp
```

```
## 4  5 11783 6.4756
```

```r
coef(object2)[5,]
```

```
##        zn     indus      chas       nox        rm       age       dis
##  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000 -0.1915697
##       rad       tax   ptratio     black     lstat      medv
##  0.5256421  0.0000000  0.0000000  0.0000000  0.0160489 -0.1140040
```

b) Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure that you are evaluating model performance using validation set error, cross- validation, or some other reasonable alternative, as opposed to using training error.

SOLUTION:

The test MSE for ridge is 37.0696, with a lambda chosen by CV of roughly 0.92.

The test MSE for lasso is 38.1394, with a lambda chosen by CV of roughly 0.169 using glmnet.

We can look a little bit more at some of the lars solutions with some other functions. The Cp statisics from the training data indicate we want the 5th solutions for both lasso and lars. There are functions that can make predictions and pull out the coefficients for those steps, or for particular lambdas, etc. The steps are most clearly seen in the summary of the object. You can do partial steps too.

```r
lasso_pred <- predict(object1, newx = xBos[test_index,], s= 4, mode = "step")
predict(object1, type = "coefficients", s= 4, mode = "step")
```

```
## $s
## [1] 4
##
## $fraction
## [1] 0.2307692
##
## $mode
## [1] "step"
##
## $coefficients
##         zn      indus       chas        nox         rm        age
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
##        dis        rad        tax    ptratio      black      lstat
##  0.00000000  0.49772708  0.00000000  0.00000000  0.00000000  0.02183975
##       medv
## -0.06298399
```

```r
mean((lasso_pred$fit - yBos.test)^2)
```

```
## [1] 39.87319
```

The test MSE is 39.8732 with the lasso fit from lars.

The lars fit is exactly the same in this case:

```r
lars_pred <- predict(object2, newx = xBos[test_index,], s= 4, mode = "step")
#predict(object2, type = "coefficients", s= 11, mode = "step")
mean((lars_pred$fit - yBos.test)^2)
```

```
## [1] 39.87319
```

These all look very comparable as solutions - the test MSEs don’t differ very much. We used a step increment for the lasso solution from lars, which is different than choosing by cv, but you could check out cv.lars for its options.

Based on these, the lowest test MSE is from the ridge fit by glmnet, so we'd propose that model.

(c) Does your chosen model involve all of the features in the data set? Why or why not?

This will depend on what model you chose. Ridge cannot set coefs to 0, but Lasso can. For example, we can compare:

```r
predict(ridgeBos.mod, type = "coefficients", s = bestlamBos)[1:14,]
```

```
##   (Intercept)            zn         indus          chas           nox
##  5.5716002575  0.0333394098 -0.0713980874 -1.2060875906 -3.5101599562
##            rm           age           dis           rad           tax
##  0.2692659199  0.0005949055 -0.6935933659  0.4013525645  0.0065558070
##       ptratio         black         lstat          medv
## -0.0844368959 -0.0034696313  0.0780256120 -0.1454922702
```

```r
predict(lassoBos.mod, type = "coefficients", s = bestlamBosl)[1:14,]
```

```
##   (Intercept)            zn         indus          chas           nox
##  4.0243228173  0.0266213396  0.0000000000 -0.9628083324  0.0000000000
##            rm           age           dis           rad           tax
##  0.0000000000  0.0000000000 -0.4846309202  0.5296616452  0.0000000000
##       ptratio         black         lstat          medv
## -0.0277282406 -0.0007388868  0.0130952796 -0.1430847333
```

Ridge won't set any coefs to 0, but if we had selected a lasso model it could set some coefficients to 0.