Appendix R7A: Dimensionality - Regularization

Ridge and LASSO Regressions

J. Alberto Espinosa

8/1/2021

Table of Contents

Technical Note: All procedures involving machine learning and cross-validation are based on random sampling. Therefore, results may change when you run the analysis multiple times, depending on the kind of random number generator **RNGkind** and the seed selected. The results in this script may not match the outputs in the main text of the book. Follow the chapter and this script appendix independently. Let's first set the random number generator default

```
RNGkind(sample.kind = "default") # To use the R default RNG
```

Dimensionality

The issue of dimensionality is often referred to as **curse of dimensionality**. As you add more predictors and complexity to a model specification, it will generally fit the data better and most fit statistics will improve. In some cases, the fit may even be perfect. For example, if you grow a tree until it has the same number of branches as you have data points, the fit will be perfect because the model will touch every single data point. Similarly, if you fit a smoothing spline model, you can set your parameters so that the resulting regression model with a jagged line that touches every single point in the data set. However, the added complexity and size of the model will cause multi-collinearity to become severe.

Naturally, these are major concerns in predictive modeling because dimensionality causes the model to be **over-fitting**, which will in turn make fit statistics like R-squared, MSE and 2LL to improve **artificially**, given the false illusion that the model is improving. However, if we CV test the model with different data, over-fitted models will have higher variance, and these predictions will not be as accurate.

Over-fitting and multi-collinearity are two of the many problems of dimensionality. The methods presented in this section are ideally suited to address dimensionality issues. There are many types of methods that address and correct for dimensionality. The 2 types of methods we will cover in this class are: (1) Regularization or Shrinkage (covered in this script); and (2) Dimension Reduction (covered in the PCR & PLS script).

Regularization (i.e., Shrinkage or Penalized) Methods.

These three names refer to the same method in which coefficients are artificially shrunk to reduce multi-collinearity. Generally speaking, Ridge and LASSO work best when the number of predictors is quite large and the data suffers from severe multi-collinearity.

When you shrink the coefficients, you are actually **biasing** the coefficients, but on the upside, shrinkage reduces the model variance. The shrinkage factor lambda is a **tuning parameter**, meaning that you can shrink coefficients to various degrees and then select the shrinkage that yields the best cross-validation testing results (i.e., best predictive accuracy and lowest variance). In this section I describe two popular regularization approaches: **Ridge** and **LASSO** (Least Absolute Shrinkage and Selection Operator) regressions. They both aim to shrink the coefficients, such that the coefficients of the less important variables become very small and, therefore, have little influence on the predictions. The magnitude of the shrinkage can be controlled with a tuning parameter called "shinkage factor" or simply "Lambda". When **Lambda = 0**, both Ridge and LASSO results are identical to OLS or GLM. When **Lambda = infinite**, both methods yielde the same results as the **Null** model. The difference between Ridge and LASSO is that as Lambda grows, Ridge coefficients become smaller, but not 0, except when Lambda is infinite. In contrast, LASSO shrinks coefficients to 0, dropping their respective predictors from the model as Lambda grows.

Because Lambda causes the coefficients to shrink, the larger the Lambda, the more **biased** they are compared to the OLS or GLM coefficients. Generally, as the Lambda grows, Ridge and LASSO improve the model's predicctive accuracy, but up to a point, after which the predictive accuracy declines. An optimal model is one with a Lambda value that minimizes the model's CV test deviance. But if the goal is interpretability, it is OK to select a model with a smaller Lambda with an acceptable CV test deviance. If you inspect the resulting coefficients, the closer they are in value to the OLS coefficients, the less biased the model and, therefore, the more interpretable.

Ridge Regression

Let's run a Ridge regression with several Lambdas and find the best Lambda that minimizes the 10FCV MSE. We will be using the **glmnet()** function from the **{glmnet}** R package. Let's start by loading the necessary packages.

```
library(glmnet)
library(ISLR) # Contains the Hitters baseball player salary data set

# This data set has several missing values, let's omit them
Hitters <- na.omit(Hitters)

# Also, let's minimize use of scientific notation
options(scipen = 4)
```

Technical Note: the **glmnet()** function has a different syntax than other functions like **lm()** and **glm()**. It requires defining an **X matrix** (of the predictors) and a **Y** vector (of the response values). That is, rather than using the familiar `y ~ x1 + x2 etc.` formula syntax, we will use the `(X, Y, etc)` syntax. We will use the **model.matrix()** function in the **{stats}** package to create the model's predictor matrix. Also, we need to remove all categorical variables because **glmnet()** only takes numerical data.

** Technical Note:** Notice below that I added [,-1] at the end of the **model.matrix()** function. This is necessary because the **model.matrix()** function will include a column full of 1's as the first column, which represents the intercept. But Ridge also renders an intercept. So, if you don't remove this first column of 1's, the Ridge model will have 2 intercepts and the coefficients will be slightly off.

Let's start by creating the **x** predictor matrix and the **y** outcome vector:

```
x <- model.matrix(Salary ~ ., data = Hitters)[,-1]
x[1:10, 1:10] # Take a look at the first 10 rows and columns

##                    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
## -Argenis Salazar     298   73     0   24  24     7     3    509   108      0
## -Andres Thomas       323   81     6   26  32     8     2    341    86      6
## -Andre Thornton      401   92    17   49  66    65    13   5206  1332    253
## -Alan Trammell       574  159    21  107  75    59    10   4631  1300     90

# You can try x without the [,-1] to the 1's on the first column

y <- Hitters$Salary # y vector with just the outcome variable
y[1:10] # Take a look at the first 10 values
```

```
## [1]  475.000  480.000  500.000   91.500  750.000   70.000  100.000   75.000
## [9] 1100.000  517.143
```

The **glmnet()** is used to fit, Ridge, LASSO and Elastic Net model. It uses the parameter `alpha =` to indicate the model to fit, with a value of **0** for a **Ridge Regression**, **1** for **LASSO** and any value between **0-1** for Elastic Net, with the value being the proportional weight given to **LASSO** (and **1 - alpha** to **Ridge**).

Technical Note: By defalut, glmnet() estimates models with 100 Lambda values, from very large to very small (in descending order). You can estimate less or more Lambda values with the `nlambda =` parameter)

```
options(scipen=999) # Disable scientific notation
ridge.mod <- glmnet(x, y, alpha = 0)
```

Note About GLM Models: To fit a Ridge regression with a binomial outcome (e.g., Logistic), use the parameter `family="binomial"`. For count outcomes, use the parameter `family="poisson"`. The interpretation of the resulting coefficients is the same as with Logistic and Count data models.

Let's take a quick look at the first 10 Lambdas and their Deviance Ratios (i.e., proportion of Null deviance explained, like an R Squared), rounded to decimals). The Lambdas are sorted from largest (close to the Null model) to smallest (close to OLS). Naturally, the explained variance of the model (relative to the Null model) increases as lambda decreases because we are getting closer to the OLS model. But this will change once we CV test the model.

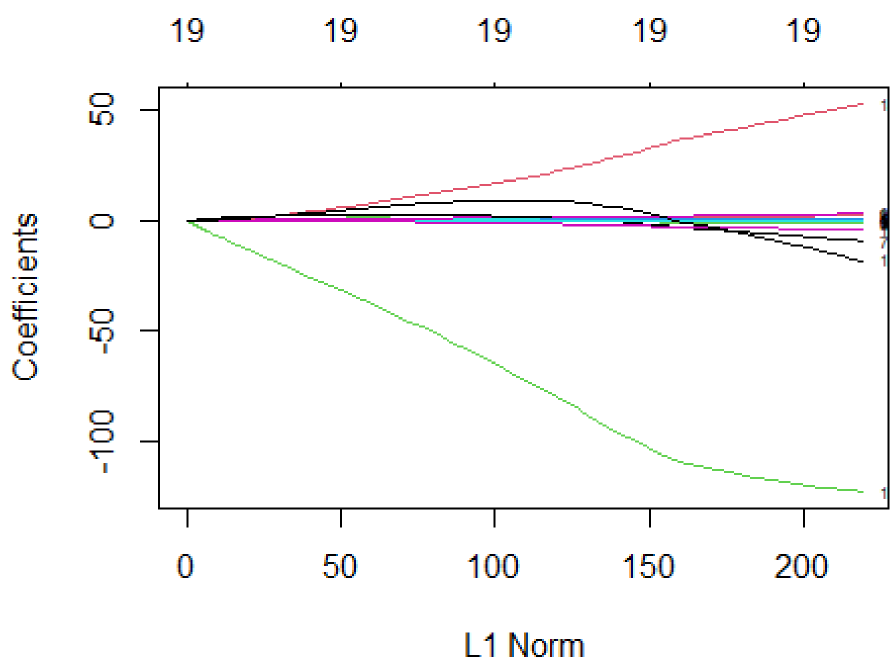
```
round(cbind("Lambda" = ridge.mod$lambda,
            "Deviance Ratio" = ridge.mod$dev.ratio)[1:10, ],
      digits=4)

##         Lambda Deviance Ratio
##  [1,] 255282.1         0.0000
##  [2,] 232603.5         0.0116
##  [3,] 211939.7         0.0127
##  [4,] 193111.5         0.0139
##  [5,] 175956.0         0.0153
##  [6,] 160324.6         0.0167
##  [7,] 146081.8         0.0183
##  [8,] 133104.3         0.0200
##  [9,] 121279.7         0.0219
## [10,] 110505.5         0.0239
```

Note: Ridge regression standardizes the coefficients because the scale of the variables can affect the proportion of shrinkage across coefficiets. However, Ridge regression can be run without standardizing coefficients simply by specifying the parameter `standardize = FALSE`. The default is TRUE.

L2 Norm

L2 Norm is a measure of the overall shrinkage of the model. It is the square root of the sum of the squared coefficients in a Ridge regression. As Lambda increases, all coefficients shrink, so the L2 Norm decreases too. **L2 Norm = square root of the sum of the model's squared coefficients.** L2 Norm is mostly usual as a preliminary visual analysis of the shrinkage patter not the model. For example, notice in the graph below shrinks the coefficients. Some get pretty small relatively quickly, but no coefficient becomes 0 until Lambda is infinitely large. Also notice how some coefficients change in size more than others and some even change signs, providing some evidence of bias. Also, notice the 19's at the top of the graph. This means that the model has 19 predictors at every shrinkage level. This will not be the case with LASSO.

```
# Note: the x axis is Labeled L1 Norm, but the prefix L1 is generally used fo
r LASSO.
plot(ridge.mod, label=T)
```



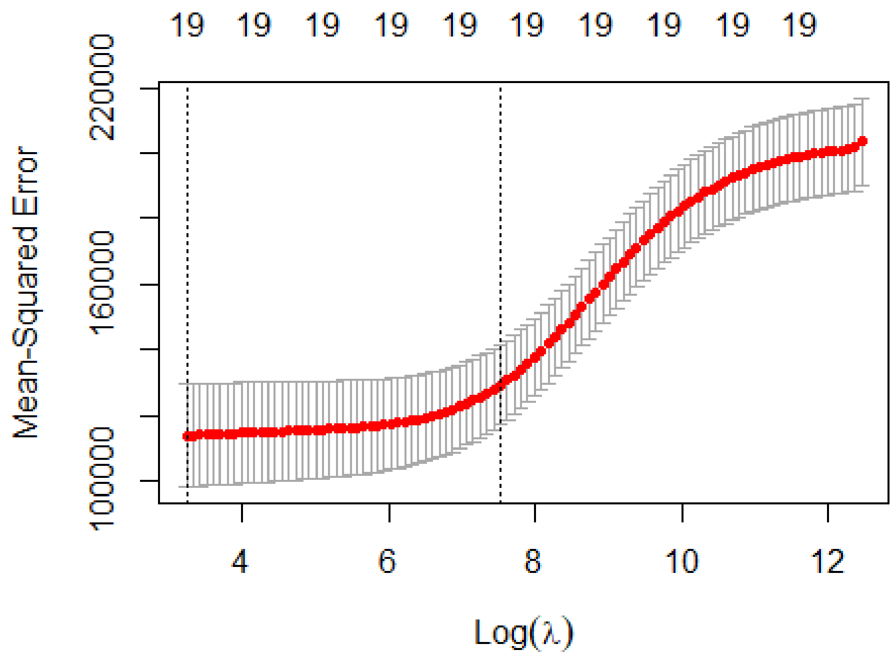
To compute the L2 Norm for any lambda, for example 50 (notice we remove the first coefficient, which is the intercept):

```
l2.norm.50 <- sqrt(sum(coef(ridge.mod)[-1, 50] ^ 2))
l2.norm.50

## [1] 22.53415
```

Optimal Shrinkage with CV Testing

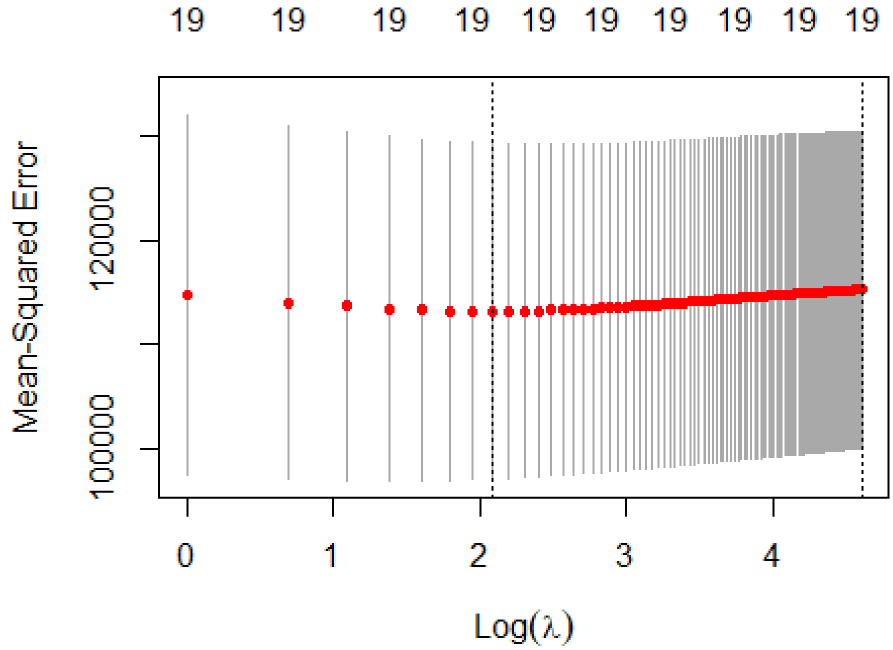
Let's find the best Lambda shrinkage value that minimizes the CV test MSE using the **cv.glmnet()** in the **{glmnet}** R package. The syntax in the cv.glmnet() function is the same as for the glmnet() function, except that it has an additional parameters, `nfolds=` to change the CV test default from **10FCV** to other number of folds.

```
set.seed(1) # Arbitrary,to get repeatable results

cv.10Fold <- cv.glmnet(x, y, alpha = 0) # 10FCV is the default

# First 10 Lambdas (remove the index to print all lambdas)
cbind("Lambda" = cv.10Fold$lambda, "10FCV MSE" = cv.10Fold$cvm)[1:10,]

##          Lambda 10FCV MSE
##  [1,] 255282.1  203414.3
##  [2,] 232603.5  201753.4
##  [3,] 211939.7  201122.4
##  [4,] 193111.5  200881.5
##  [5,] 175956.0  200618.5
##  [6,] 160324.6  200331.2
##  [7,] 146081.8  200017.6
##  [8,] 133104.3  199675.6
##  [9,] 121279.7  199302.8
## [10,] 110505.5  198896.5

# Let's plot all the Lambdas against their 10FCV MSE
plot(cv.10Fold)
```



The plot suggests that the MSE declines as the shrinkage lambda decreases. There is no clearly visible minimum in the graph. But we can evaluate this more carefully, by exploring several small lambdas, say from 1 to 100.

```
set.seed(1)
cv.10Fold <- cv.glmnet(x, y, alpha = 0, lambda = seq(0, 100)) # 10FCV is the
default
plot(cv.10Fold)
```



The new plot shows some possible minimum value for the MSE around log(lambda) = 2. Let's find it more precisely, quantitatively. The best best Lambda is the one that minimizes the 10FCV test MSE. The `cv.glmnet()` output object has the attribute $lambda.min ** that(conveniently)storestheLambdawiththesmallestCVtestMSE.Let'sdisplayit, alongwithitslogtoma$ *****cvm** (has all MSE's for all Lambdas) and finding its minimum value. I then display the results. And, sure enough, the best lambda is 8 with log(lambda) = 2.079 and 10FCV = 113,198.6.

```
best.lambda <- cv.10Fold$lambda.min
min.mse <- min(cv.10Fold$cvm)

cbind("Best Lambda"=best.lambda,
      "Log(Lambda)"=log(best.lambda),
      "Best 10FCV MSE" = min.mse)

##      Best Lambda Log(Lambda) Best 10FCV MSE
## [1,]           8    2.079442       113198.6
```

Extracting Ridge Coefficients

You can extract Ridge regression coefficients for any level of Lambda shrinkage with the `coef()` function and the `s=` parameter:

```
options(scipen = 4)
# Same as OLS
ridge.0 <- round(coef(ridge.mod, s = 0), digits = 4)
```

```r
# For the optimal Lambda
ridge.best <- round(coef(ridge.mod, s = best.lambda), digits = 4)

# Some arbitrary Lambda = 50
ridge.50 <- round(coef(ridge.mod, s = 50), digits = 4)

# Extreme shrinkage, close to the Null model
ridge.huge <- round(coef(ridge.mod, s = 10 ^ 20), digits = 4)

ridge.all <- cbind(ridge.0, ridge.best, ridge.50, ridge.huge)
colnames(ridge.all) <-
    c("OLS, Lambda = 0", "Best Lambda = 8", "Lambda = 50", "Almost Null")

ridge.all # Display all coefficients

## 20 x 4 sparse Matrix of class "dgCMatrix"
##               OLS, Lambda = 0 Best Lambda = 8 Lambda = 50 Almost Null
## (Intercept)           81.1269         81.1269     48.2165    535.9259
## AtBat                 -0.6816         -0.6816     -0.3539      0.0000
## Hits                   2.7723          2.7723      1.9532      0.0000
## HmRun                 -1.3657         -1.3657     -1.2851      0.0000
## Runs                   1.0148          1.0148      1.1563      0.0000
## RBI                    0.7130          0.7130      0.8088      0.0000
## Walks                  3.3786          3.3786      2.7098      0.0000
## Years                 -9.0668         -9.0668     -6.2029      0.0000
## CAtBat                -0.0012         -0.0012      0.0061      0.0000
## CHits                  0.1361          0.1361      0.1071      0.0000
## CHmRun                 0.6980          0.6980      0.6291      0.0000
## CRuns                  0.2959          0.2959      0.2173      0.0000
## CRBI                   0.2571          0.2571      0.2153      0.0000
## CWalks                -0.2790         -0.2790     -0.1489      0.0000
## LeagueN               53.2127         53.2127     45.8626      0.0000
## DivisionW           -122.8345       -122.8345   -118.2304      0.0000
## PutOuts                0.2639          0.2639      0.2502      0.0000
## Assists                0.1699          0.1699      0.1208      0.0000
## Errors                -3.6856         -3.6856     -3.2771      0.0000
## NewLeagueN           -18.1051        -18.1051     -9.4235      0.0000
```

As you can see from the output above, the OLS coefficients are the same as those for the Ridge model with Lambda = 8 (minimal shrinkage). The coefficients appear to be identical, but that is due to rounding. With sufficient decimals you should see some very small differences. You can also see that an arbitrary value of Lambda = 50 biases the coefficients a little, but not by much. In contrast, the large Lambda yields an almost Null model. I say "almost" because Ridge does not drop the coefficients, and with sufficient decimals you would see very small non-zero coefficients. But for practical reasons, a Lambda = 10 ^ 20 yields very similar results to the Null model.

You can also get coefficients for a specific group of lambdas, any sequence of lambdas or a particular lambda number:

```
# Coefficients for a group of Lambdas

ridge.mod.group <- glmnet(x, y, alpha = 0,
                          lambda = c(10^20, 10000, 10, 0))
round(coef(ridge.mod.group), digits = 4) # Take a look

## 20 x 4 sparse Matrix of class "dgCMatrix"
##                   s0       s1        s2        s3
## (Intercept) 535.9259 392.7312  122.3737  161.9747
## AtBat         0.0000   0.0411   -1.2068   -1.9484
## Hits          0.0000   0.1545    4.2932    7.3497
## HmRun         0.0000   0.5814   -0.4841    4.1744
## Runs          0.0000   0.2574    0.4559   -2.2543
## RBI           0.0000   0.2670    0.3972   -1.0056
## Walks         0.0000   0.3236    4.4015    6.1740
## Years         0.0000   1.2262  -10.7809   -2.8965
## CAtBat        0.0000   0.0035   -0.0226   -0.1878
## CHits         0.0000   0.0130    0.1652    0.2158
## CHmRun        0.0000   0.0974    0.6898   -0.0523
## CRuns         0.0000   0.0260    0.4727    1.4041
## CRBI          0.0000   0.0269    0.3338    0.7620
## CWalks        0.0000   0.0278   -0.4597   -0.7913
## LeagueN       0.0000   0.1688   58.9833   63.5176
## DivisionW     0.0000  -7.0644 -124.1991 -116.7595
## PutOuts       0.0000   0.0186    0.2742    0.2814
## Assists       0.0000   0.0029    0.2369    0.3767
## Errors        0.0000  -0.0245   -3.8505   -3.4180
## NewLeagueN    0.0000   0.3930  -25.4166  -25.6552

# Coefficients for a sequence of Lambdas

ridge.mod.sequence <- glmnet(x, y, alpha = 0, lambda = seq(1, 5))
round(coef(ridge.mod.sequence), digits = 4) # Take a look

## 20 x 5 sparse Matrix of class "dgCMatrix"
##                    s0        s1        s2        s3        s4
## (Intercept)  143.9106  150.3870  154.8083  159.1816  162.4634
## AtBat         -1.5387   -1.6311   -1.7190   -1.8126   -1.9006
## Hits           5.4047    5.6976    6.0451    6.4387    6.8769
## HmRun          0.5463    0.7278    1.1689    1.7513    2.6002
## Runs          -0.1905   -0.3778   -0.6670   -1.0199   -1.5030
## RBI            0.0863    0.0408   -0.0849   -0.2531   -0.5051
## Walks          5.0681    5.2665    5.4616    5.6794    5.9210
## Years        -10.3670  -10.3179   -9.5292   -8.4591   -6.5952
## CAtBat        -0.0495   -0.0580   -0.0739   -0.0942   -0.1270
## CHits          0.1815    0.2031    0.2098    0.2159    0.2236
## CHmRun         0.6451    0.7134    0.6671    0.5850    0.4177
```

```
## CRuns          0.6622    0.7101    0.8042    0.9205    1.0939
## CRBI           0.3905    0.3705    0.4008    0.4474    0.5344
## CWalks        -0.5767   -0.6132   -0.6492   -0.6891   -0.7358
## LeagueN       60.8330   61.4366   61.8941   62.3310   62.8899
## DivisionW   -123.1024 -122.6512 -121.8053 -120.7689 -119.2272
## PutOuts        0.2783    0.2796    0.2806    0.2815    0.2820
## Assists        0.2799    0.2926    0.3072    0.3240    0.3458
## Errors        -3.7777   -3.7555   -3.7034   -3.6377   -3.5495
## NewLeagueN   -27.3382  -27.9432  -27.9504  -27.7946  -27.2580

# Coefficients for a particular sequence order of Lambda

ridge.mod.sequence$lambda[3] # Display the third Lambda

## [1] 3

round(coef(ridge.mod.sequence)[,3], digits = 4) # Its coefficients

## (Intercept)       AtBat        Hits       HmRun        Runs         RBI
##    154.8083     -1.7190      6.0451      1.1689     -0.6670     -0.0849
##       Walks       Years      CAtBat       CHits      CHmRun       CRuns
##      5.4616     -9.5292     -0.0739      0.2098      0.6671      0.8042
##        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists
##      0.4008     -0.6492     61.8941   -121.8053      0.2806      0.3072
##      Errors  NewLeagueN
##     -3.7034    -27.9504
```

Predicting with Ridge

We can use the **predict()** function to obtain Ridge regression coefficients using the attribute `type = "coefficients"`. Let's find the Ridge regression coefficients for the best lambda:

```
predict(ridge.mod, s = best.lambda, type = "coefficients")

## 20 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)   81.126931475
## AtBat         -0.681595884
## Hits           2.772311573
## HmRun         -1.365680118
## Runs           1.014826485
## RBI            0.713022451
## Walks          3.378557588
## Years         -9.066800376
## CAtBat        -0.001199478
## CHits          0.136102881
## CHmRun         0.697995815
## CRuns          0.295889601
## CRBI           0.257071062
## CWalks        -0.278966594
```

```
## LeagueN       53.212720264
## DivisionW   -122.834451470
## PutOuts        0.263887567
## Assists        0.169879574
## Errors        -3.685645334
## NewLeagueN   -18.105095858
```

IMPORTANT: with shrinkage methods like Ridge regressions, p-values no longer relevant, because all coefficients are retained in the model, although shrunk (i.e., biased). The actual magnitude of the coefficients tells you how important the corresponding variable is in the prediction. Similarly, the R-squared values are not reported, but for quantitative outcome models, you can use the deviance ratio illustrated above as an R squared.

Now let's use the **predict()** function to make actual predictions. To do predictions with Ridge regression, you need to store the observations you want to do predictions for in a **new X** and specify it with the **newx=** parameter. This new X can either be new data for which you want to make predictions, or some test subsample to evaluate the model's accuracy. For the purposes of this example, let's just draw a random test subsample with 5% of the existing observations, using the best lambda we found above from the same **x** predictor matrix above.

```
set.seed(1)
test <- sample(1:nrow(x), 0.05 * nrow(x))
x.test <- x[test,] # Extract test sub-sample

ridge.pred <- predict(ridge.mod, s = best.lambda, newx = x.test)
ridge.pred # Take a look
```

```
##                         1
## -Mike Heath      463.6768
## -John Russell    385.6552
## -Pete Incaviglia 307.2966
## -Glenn Davis     776.5389
## -Frank White     859.7506
## -Rafael Santana  347.9036
## -Chili Davis     680.8958
## -Herm Winningham 213.7646
## -Robby Thompson  390.6993
## -Joe Carter      647.9247
## -Spike Owen      344.9689
## -Mike Easler     673.2900
## -Chris Bando     320.4533
```

Ridge and Logistic Regression

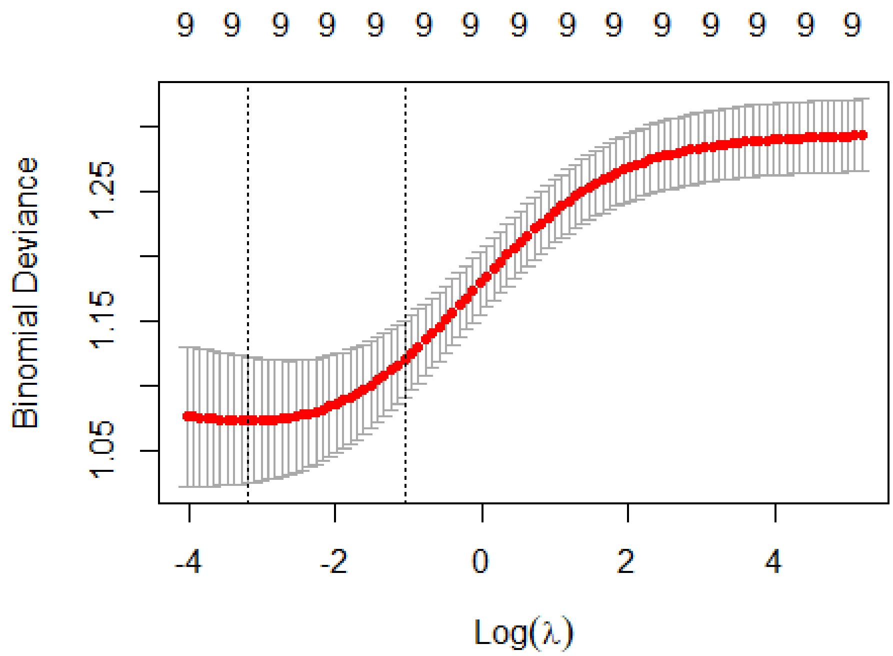
You can fit a Ridge shrinkage model on a Logistic Regression simply by adding the parameter `family = "binomial"` to both functions, **glmnet()** and **cv.glmnet()**. Selecting the best lambda is identical to quantitative Ridge models. The coefficient interpretation in Ridge Logistic is the same as for Logistic Regression interpretation based on log-odds effects. To illustrate Ridge Logistic, we can access the SAHeart.data dataset in Prof. Robert Tibshirani's web site, which contains predictors for coronary heart disease (binary outcome, chd = 1 for disease, 0 otherwise). We then create the x predictor matrix and y outcome vector in the same way we did before.

```
heart <- read.table("http://www-stat.stanford.edu/~tibs/ElemStatLearn/dataset
s/SAheart.data", sep=",",head=T,row.names=1)

x <- model.matrix(chd ~ .,data=heart)[,-1]
y <- heart$chd
```

We then follow the same steps as for quantitative Ridge regressions, described above, with three differences:

- We use the `family = "binomial"` parameter
- The CV test reported in **$cvm** is deviance using the `2LL = -2 * Log-Likelihood` fit statistic for GLM models
- If we want to use **classification error** as the CV test deviance, we simply include the `measure.type = "class"` parameter.

```
# Fit the Ridge Logistic model

ridge.logit=glmnet(x, y, alpha = 0, family = "binomial")

# Find best lambda:

set.seed(1)
cv.10Fold.logit <- cv.glmnet(x, y, alpha = 0, family = "binomial")

# Report (first 10 Lambdas to minimize the printout) and plot

cbind("Lambda" = cv.10Fold.logit$lambda,
      "10-Fold CV Deviance" = cv.10Fold.logit$cvm)[1:10,]

##          Lambda 10-Fold CV Deviance
##  [1,] 177.45951            1.294554
##  [2,] 161.69449            1.293559
##  [3,] 147.33000            1.293179
##  [4,] 134.24161            1.293039
##  [5,] 122.31596            1.292886
##  [6,] 111.44974            1.292717
##  [7,] 101.54886            1.292533
##  [8,]  92.52753            1.292331
##  [9,]  84.30764            1.292109
## [10,]  76.81798            1.291866

plot(cv.10Fold.logit)
```



Now let's find the **Best Lambda**.

```
best.lambda.logit <- cv.10Fold.logit$lambda.min
log(best.lambda) # Log to spot it in the plot

## [1] 2.079442

min.mse.logit <- min(cv.10Fold.logit$cvm)

cbind("Best Lambda" = best.lambda.logit,
      "Log(Lambda)" = log(best.lambda.logit),
      "Best 10FCV Deviance" = min.mse.logit)

##      Best Lambda Log(Lambda) Best 10FCV Deviance
## [1,]  0.04099545   -3.194294            1.073231
```

Then let's display coefficients

```
# Logit Model
ridge.logit.coef.0 <- coef(ridge.logit, s = 0)

# Alternatively, you can use
# predict(ridge.logit, s=best.lambda.logit, type="coefficients")

# Best Lambda Model
ridge.logit.coef.best <- coef(ridge.logit, s=best.lambda.logit)
```

```r
# Almost Null Model
ridge.logit.coef.large <- coef(ridge.logit, s = 10 ^ 20)

# Output results

all.coefs <- round(cbind(ridge.logit.coef.0,
                         ridge.logit.coef.best,
                         ridge.logit.coef.large), digits = 4)

colnames(all.coefs) <- c("Logistic", "Best Lambda", "Almost Null Model")
all.coefs

## 10 x 3 sparse Matrix of class "dgCMatrix"
##                 Logistic Best Lambda Almost Null Model
## (Intercept)      -5.7222     -5.3187           -0.6353
## sbp               0.0065      0.0063            0.0000
## tobacco           0.0754      0.0705            0.0000
## ldl               0.1575      0.1421            0.0000
## adiposity         0.0174      0.0167            0.0000
## famhistPresent    0.8462      0.7664            0.0000
## typea             0.0334      0.0281            0.0000
## obesity          -0.0491     -0.0376            0.0000
## alcohol           0.0002      0.0004            0.0000
## age               0.0391      0.0339            0.0000
```

LASSO Regression

To fit LASSO regression models we follow the same steps as above, except that we use the parameter `alpha = 1` rather than 0.

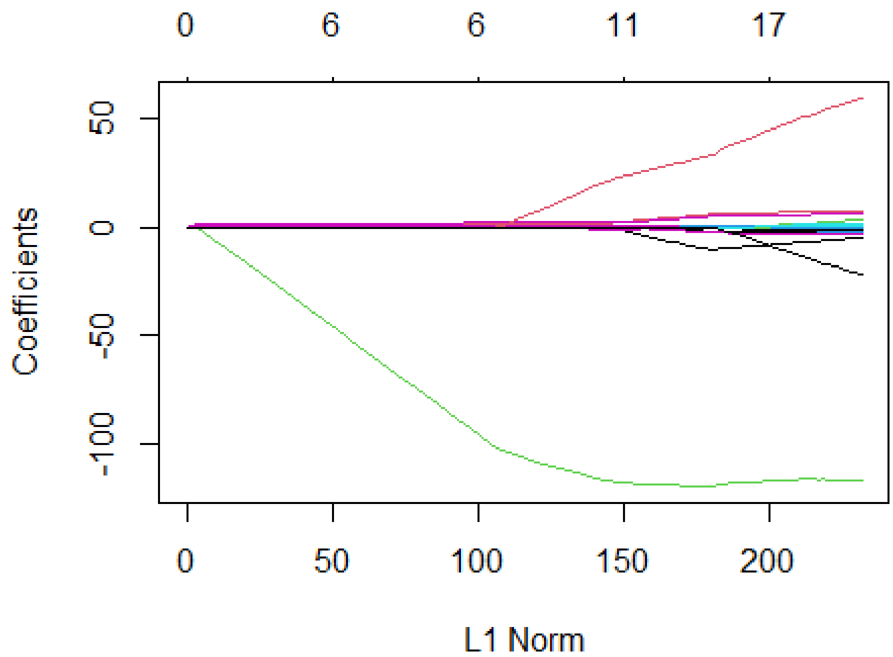
We can also fit Elastic Net regression models using `0 < alpha < 1`, which is a hybrid between Ridge and LASSO.

Again, the one difference between Ridge regression and LASSO is that coefficents can shrink significantly with Ridge regression, but they never become 0 (except when lambda is infinite). In contrasts, some LASSO coefficients can shrink to 0. The more you shrink, the more coefficients drop out of the model. Let's re-create the x predictor matrix and y outcome vector for the Hitters data set.

```r
x <- model.matrix(Salary ~ .,data=Hitters)[,-1]
y <- Hitters$Salary
```

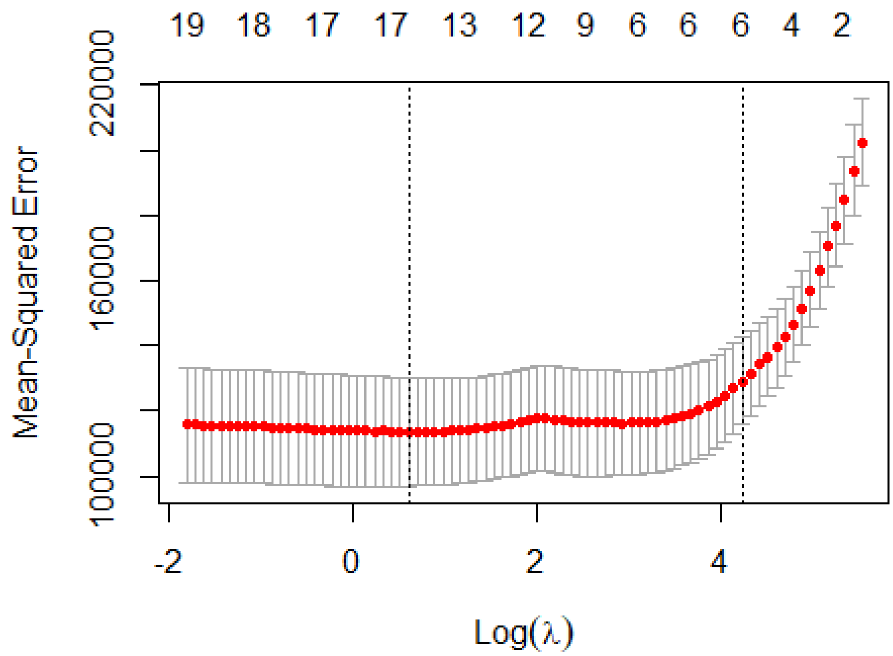
Now let's fit the LASSO model and graph the **L1 Norm**. L1 Norm is the sum of the absolute values of the coefficients in a LASSO regression. Like L2 Norm, it measures the total amount of shrinkage obtained by a particular Lambda value used, but using absolute values, rather than squared values. The larger the L1 the smaller the shrinkage.

```r
lasso.mod <- glmnet(x, y, alpha=1)
plot(lasso.mod) # Plot the L1 Norm against the coefficients
```



Optimal LASSO Lambda

```
set.seed(1) # To get repeatable results
cv.10Fold.lasso <- cv.glmnet(x, y, alpha=1) # 10-Fold is the default
# Use the parameter nfolds=xx to change the number of folds

# List all lambdas and corresponding CV test deviance (2LL)

round(cbind("Lambda" = cv.10Fold.lasso$lambda,
            "10 Fold 2LL" = cv.10Fold.lasso$cvm), digits = 4)[1:10,]

##          Lambda 10 Fold 2LL
##  [1,] 255.2821    202546.7
##  [2,] 232.6035    193851.4
##  [3,] 211.9397    184575.3
##  [4,] 193.1115    177082.0
##  [5,] 175.9560    170421.3
##  [6,] 160.3246    163395.7
##  [7,] 146.0818    157050.7
##  [8,] 133.1043    151398.1
##  [9,] 121.2797    146527.1
## [10,] 110.5055    142542.5

plot(cv.10Fold.lasso) # Plot all lambdas vs. MSEs
```



Let's find the best lambda that minimizes 10FCV deviance

```
best.lambda.lasso <- cv.10Fold.lasso$lambda.min
log(best.lambda.lasso) # Spot it in the plot

## [1] 0.6115809

min.mse.lasso <- min(cv.10Fold.lasso$cvm)

round(cbind("Best Lambda"=best.lambda.lasso,
            "Log(Lambda)"=log(best.lambda.lasso),
            "Best 10FCV MSE" = min.mse.lasso), digits = 4)

##      Best Lambda Log(Lambda) Best 10FCV MSE
## [1,]      1.8433      0.6116       113581.6
```

Extracting LASSO Coefficients

```
lasso.0 <- coef(lasso.mod, s = 0) # No shrinkage, same as OLS
lasso.best <- coef(lasso.mod, s = best.lambda.lasso) # best Lambda
lasso.50 <- coef(lasso.mod, s = 50) # Some lambda=50
lasso.large <- coef(lasso.mod, s = 10^20) # Null model

lasso.all <- cbind(ridge.0, ridge.best, ridge.50, ridge.huge)
colnames(lasso.all) <-
    c("OLS, Lambda = 0", "Best Lambda = 1.84", "Lambda = 50", "Null")
```

```r
lasso.all # Display all coefficients

## 20 x 4 sparse Matrix of class "dgCMatrix"
##             OLS, Lambda = 0 Best Lambda = 1.84 Lambda = 50     Null
## (Intercept)         81.1269            81.1269     48.2165 535.9259
## AtBat               -0.6816            -0.6816     -0.3539   0.0000
## Hits                 2.7723             2.7723      1.9532   0.0000
## HmRun               -1.3657            -1.3657     -1.2851   0.0000
## Runs                 1.0148             1.0148      1.1563   0.0000
## RBI                  0.7130             0.7130      0.8088   0.0000
## Walks                3.3786             3.3786      2.7098   0.0000
## Years               -9.0668            -9.0668     -6.2029   0.0000
## CAtBat              -0.0012            -0.0012      0.0061   0.0000
## CHits                0.1361             0.1361      0.1071   0.0000
## CHmRun               0.6980             0.6980      0.6291   0.0000
## CRuns                0.2959             0.2959      0.2173   0.0000
## CRBI                 0.2571             0.2571      0.2153   0.0000
## CWalks              -0.2790            -0.2790     -0.1489   0.0000
## LeagueN             53.2127            53.2127     45.8626   0.0000
## DivisionW         -122.8345          -122.8345   -118.2304   0.0000
## PutOuts              0.2639             0.2639      0.2502   0.0000
## Assists              0.1699             0.1699      0.1208   0.0000
## Errors              -3.6856            -3.6856     -3.2771   0.0000
## NewLeagueN         -18.1051           -18.1051     -9.4235   0.0000
```

Notice again that the OLS and best LASSO models are just about the same. This is not surprising because the best Ridge model was also very close to the OLS model, suggesting that there are no major dimensionality issues, so naturally, the best LASSO model will be close to the OLS model.

Predicting with LASSO

We can use the `predict()` function to obtain LASSO regression coefficients

```r
lasso.coef <- predict(lasso.mod, s = best.lambda.lasso,type="coefficients")
lasso.coef # Notice that the dropped predictors show blanks

## 20 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  142.877615462
## AtBat         -1.793369129
## Hits           6.187727974
## HmRun          0.232913247
## Runs           .
## RBI            .
## Walks          5.147970764
## Years        -10.392782094
## CAtBat        -0.004469497
## CHits          .
```

```
## CHmRun          0.585358719
## CRuns           0.763882251
## CRBI            0.388422191
## CWalks         -0.629678347
## LeagueN        34.226933747
## DivisionW    -118.980715754
## PutOuts         0.279042882
## Assists         0.223985943
## Errors         -2.436300479
## NewLeagueN      .
```

To make actual predictions and to run LASSO Logistic models, the processes are identical to the ones for Ridge regression above.