R Notebook

LASSO/Ridge Regression Tutorial

Import Necessary Packages and Data

I always import tidyverse. This lets me have most of the necessary data manipulation functions I'll need. glmnet is our LASSO regression package

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

glmnet comes with a small in-built dataset that's basically just 20 random X variables and a random Y variable. We can use that for our purposes here. In general, glmnet can fit models with several different data types in R: dataframes, matrices, vectors, probably more.

```
data <- data(QuickStartExample)
x <- QuickStartExample$x
y <- QuickStartExample$y
```

Let's view the predictors (x) and the outcome (y). There are 20 variables in x:

```
head(x)
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]      [,6]
## [1,]  0.2738562 -0.0366722  0.8547269  0.9675242  1.4154898 0.5234059
## [2,]  2.2448169 -0.5460300  0.2340651 -1.3350304  1.3130758 0.5212746
## [3,] -0.1254230 -0.6068782 -0.8539217 -0.1487772 -0.6646828 0.6066164
## [4,] -0.5435734  1.1083583 -0.1042480  1.0165262  0.6999042 1.6550164
## [5,] -1.4593984 -0.2744945  0.1119060 -0.8517877  0.3152839 1.0507493
## [6,]  1.0632081 -0.7535232 -1.3825534  1.0762270  0.3700331 1.4987212
```

```
##            [,7]        [,8]       [,9]      [,10]       [,11]       [,12]
## [1,]  0.5626882  1.11122333  1.6408214  0.6187067  0.99993483 -0.07841916
## [2,] -0.6100346 -0.86139651 -0.2704635  0.2300825 -0.10570898  0.16314122
## [3,]  0.1617207 -0.86272165  0.6042102  1.1939768  0.50125094 -0.94520592
## [4,]  0.4899635  0.02338209  0.2560304 -0.1273140 -0.06262846  0.64195468
## [5,]  1.3863575  0.28450104  1.1404976  2.6813460 -1.01473386  0.36704372
## [6,] -0.3604525 -0.21421571  1.8266483  0.8711225 -0.06372928  1.00507110
##            [,13]       [,14]      [,15]      [,16]      [,17]       [,18]
## [1,] -0.60332610  0.03323168 -0.7008845  1.1578379  1.4578156  0.77490699
## [2,]  0.76207661  0.67812003 -0.5282670 -0.8791548 -0.4729134 -1.11717309
## [3,]  0.39890263 -0.76478505  1.2854019  0.6448767  0.1792455  0.04473756
## [4,]  0.07548198 -1.37846440 -1.0247390 -2.1183615 -0.4695345  0.69779616
## [5,]  1.73759745 -1.26612706  1.4519995 -0.7894756 -0.9843860 -1.63700993
## [6,]  0.31520786  0.53671846 -1.2624227 -1.5848662 -0.6314991 -1.87874350
##           [,19]      [,20]
## [1,] -1.2685177  1.9935800
## [2,] -0.7377321 -1.0787929
## [3,]  1.1053071  0.3040545
## [4,]  0.8656362 -0.7894895
## [5,] -0.5829168 -1.5289104
## [6,]  0.4504287  1.4422643
```

```
head(y)
```

```
##            [,1]
## [1,] -1.2748860
## [2,]  1.8434251
## [3,]  0.4592363
## [4,]  0.5640407
## [5,]  1.8729633
## [6,]  0.5275317
```

Train a LASSO

Now let's demonstrate how to fit a LASSO regression. The glmnet function actually estimates the model, which I assign to the object "fit". Recall from the lecture that the regularized linear regression we want to estimate for prediction is the following:

$$\arg\min_{\beta} \quad \underbrace{\sum_{i=1}^{N}(Y_i - \boldsymbol{\beta}^T \boldsymbol{X}_i)^2}_{\text{The regular OLS regression}} \quad + \quad \underbrace{\lambda(\sum_{k=1}^{K}|\beta_k|^{\alpha})^{1/\alpha}}_{\text{The regularization terms}}$$

Some important parameters:

1. α is the LASSO penalty. Set alpha = 1 for LASSO regression, alpha = 0 to estimate Ridge regressions, alphas between [0,1] correspond to elastic nets that favor LASSO or Ridge depending on the value you select
2. nlambda corresponds to the number of λ values you test for. Recall that λ is the severity of the penalty for non-zero coefficients. In other words, it's how much we shrink the coefficient estimates from the regular OLS part. So larger values of lambda mean that we will ultimately consider fewer X variables as "important". I can specify λ manually if I want, but I don't know what the correct value should be. So I let the data tell me by estimating 100 LASSOs for 100 different values of λ. It will create 100 evenly spaced lambda values between lambda.min.ratio and lambda.max (discussed next) and estimate a LASSO for each.

3. lambda.min.ratio sets the minimum value of λ. It defaults (in the normal case where there are more observations than variables) to 0.00001. I don't specify this parameter here because I'm fine with the defaults
4. lambda.max is the maximum value of λ we search over. This also defaults to the smallest value of λ where all coefficients are 0 after shrinking. That is, imposing more harsh penalties than lambda.max does nothing because we're already shrinking all coefficients to 0.

```
fit <- glmnet(x,
              y,
              alpha = 1,
              nlambda = 100
              )
```

We can look at the training results here:

1. Df corresponds to the number of non-zero (i.e. "important") X variables for the given value of λ (recall larger values of λ indicate more harsh penalties, so larger lambdas will mean fewer "important" variables)
2. %Dev is kindof like R^2. It corresponds to the share of in-sample variation that's explained by the model.
3. Note that if the %Dev doesn't change much between lambda values, it might stop short of training all LASSOs. Here it stopped after training 67 models. This is to save computation time in cases where training more models doesn't gain us much.

```
print(fit)
```

```
##
## Call:  glmnet(x = x, y = y, alpha = 1, nlambda = 100)
##
##    Df  %Dev  Lambda
## 1   0  0.00 1.63100
## 2   2  5.53 1.48600
## 3   2 14.59 1.35400
## 4   2 22.11 1.23400
## 5   2 28.36 1.12400
## 6   2 33.54 1.02400
## 7   4 39.04 0.93320
## 8   5 45.60 0.85030
## 9   5 51.54 0.77470
## 10  6 57.35 0.70590
## 11  6 62.55 0.64320
## 12  6 66.87 0.58610
## 13  6 70.46 0.53400
## 14  6 73.44 0.48660
## 15  7 76.21 0.44330
## 16  7 78.57 0.40400
## 17  7 80.53 0.36810
## 18  7 82.15 0.33540
## 19  7 83.50 0.30560
## 20  7 84.62 0.27840
## 21  7 85.55 0.25370
## 22  7 86.33 0.23120
## 23  8 87.06 0.21060
## 24  8 87.69 0.19190
```

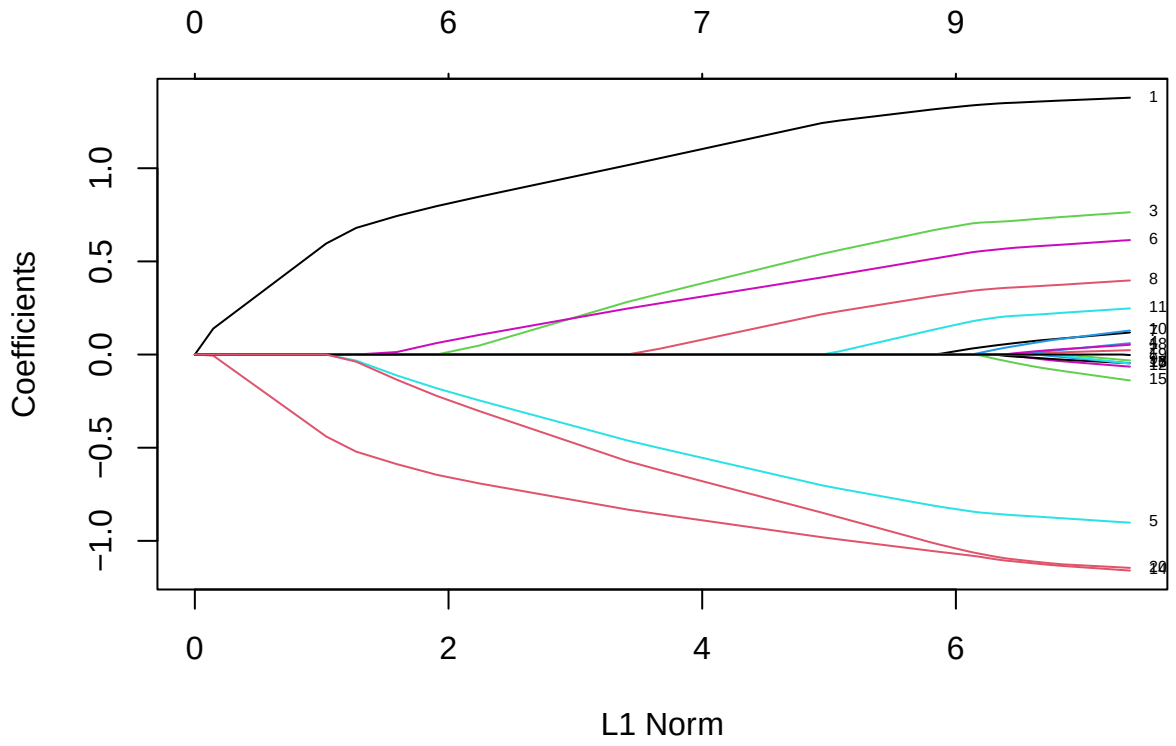
```
## 25  8 88.21 0.17490
## 26  8 88.65 0.15930
## 27  8 89.01 0.14520
## 28  8 89.31 0.13230
## 29  8 89.56 0.12050
## 30  8 89.76 0.10980
## 31  9 89.94 0.10010
## 32  9 90.10 0.09117
## 33  9 90.23 0.08307
## 34  9 90.34 0.07569
## 35 10 90.43 0.06897
## 36 11 90.53 0.06284
## 37 11 90.62 0.05726
## 38 12 90.70 0.05217
## 39 15 90.78 0.04754
## 40 16 90.86 0.04331
## 41 16 90.93 0.03947
## 42 16 90.98 0.03596
## 43 17 91.03 0.03277
## 44 17 91.07 0.02985
## 45 18 91.11 0.02720
## 46 18 91.14 0.02479
## 47 19 91.17 0.02258
## 48 19 91.20 0.02058
## 49 19 91.22 0.01875
## 50 19 91.24 0.01708
## 51 19 91.25 0.01557
## 52 19 91.26 0.01418
## 53 19 91.27 0.01292
## 54 19 91.28 0.01178
## 55 19 91.29 0.01073
## 56 19 91.29 0.00978
## 57 19 91.30 0.00891
## 58 19 91.30 0.00812
## 59 19 91.31 0.00739
## 60 19 91.31 0.00674
## 61 19 91.31 0.00614
## 62 20 91.31 0.00559
## 63 20 91.31 0.00510
## 64 20 91.31 0.00464
## 65 20 91.32 0.00423
## 66 20 91.32 0.00386
## 67 20 91.32 0.00351
```

We can also examine the coefficient estimates for each variable as λ decreases. Here's how to interpret this graph:

1. Each curve is a variable in X. They are labeled according to variable name
2. The y-axis corresponds to the coefficient estimate for each variable over different values of λ
3. The bottom x-axis corresponds to the λ. As we move left to right, the λ is decreasing. Note that in the plot function you can optionally change this to show the $\log(\lambda)$ instead of the L1 Norm
4. The top x-axis is the number of non-zero ("important") coefficients at the value of lambda on the bottom x-axis. That is, as λ becomes less strict (moving from left to right), we deem more variables as

important. This helps us see which variables were important over all regularization intensities. For example, V1 and V20 were consistently the most important predictors.

```
plot(fit,
     #xvar = "lambda",
     label = TRUE)
```



We can look at the actual coefficient estimates for each variable at any given λ value. The models I'm looking at correspond to ones closest to $\lambda = 0.1$ and $\lambda = 0.5$, which is the parameter s in the code below.

Unsurprisingly, the one with less strict regularization ($\lambda = 0.1$) estimated more non-zero coefficients.

```
coef(fit,
     s = c(0.1, 0.5))
```

```
## 21 x 2 sparse Matrix of class "dgCMatrix"
##                       s1         s2
## (Intercept)  0.150928072  0.2613110
## V1           1.320597195  1.0055473
## V2           .            .
## V3           0.675110234  0.2677134
## V4           .            .
## V5          -0.817411518 -0.4476475
## V6           0.521436671  0.2379283
## V7           0.004829335  .
## V8           0.319415917  .
## V9           .            .
## V10          .            .
```

```
## V11          0.142498519  .
## V12          .            .
## V13          .            .
## V14         -1.059978702 -0.8230865
## V15          .            .
## V16          .            .
## V17          .            .
## V18          .            .
## V19          .            .
## V20         -1.021873704 -0.5553675
```

Make predictions on new data using our LASSO

Let's generate some new out-of-sample data to make predictions on with our LASSO model. Here, I generate 5 new observations with each of our 20 predictor variables

```
set.seed(29)
nx <- matrix(rnorm(5 * 20), 5, 20)
nx
```

```
##            [,1]       [,2]        [,3]        [,4]       [,5]        [,6]
## [1,] -1.2833712  2.1104419  0.42860107 -0.34216061  0.2824026 -0.09381378
## [2,] -1.2634498 -0.5214171  1.92485609 -1.44201141  2.1983026 -0.83640257
## [3,]  0.2146729 -0.9300566  0.30992181 -0.01944062 -1.4486192  0.00307641
## [4,]  0.9469490  0.4193712 -0.03158589  1.41668616  1.7500686  0.04197496
## [5,] -1.1750246  1.1111370  0.47206596 -0.44635576  0.2527575  0.15433872
##             [,7]       [,8]       [,9]       [,10]      [,11]       [,12]
## [1,] -0.21301893 -1.186409  0.4193638  0.48025727 -1.0326186  0.06377042
## [2,]  0.15005878 -0.683786  0.9567204  0.07359607 -0.5764074 -0.33243208
## [3,]  0.03644079 -1.941015 -0.4157724 -0.63780390  1.2016426 -1.29608634
## [4,]  0.03320331  1.265440  1.7612773 -0.03802815 -0.2983949  0.52568159
## [5,] -0.19978216 -2.434234 -1.4261084  0.71690843  0.1197445  0.20987226
##           [,13]       [,14]      [,15]      [,16]      [,17]      [,18]
## [1,] -0.2590092  1.33617939  1.1042719 -2.3238624 -0.9976062 -0.3919676
## [2,]  1.0352162 -0.30519253 -1.0787079 -1.1862360 -1.1431874  0.1562070
## [3,] -0.7519736 -0.08110869  0.6253085 -1.0781804  0.9234519  0.2688388
## [4,]  0.1145837 -1.61811018 -1.6822411 -0.6234341 -0.2397104  0.7765172
## [5,] -1.4768713 -0.54197575 -0.5966446 -2.2802688 -1.3814809 -0.1544760
##           [,19]      [,20]
## [1,] -0.2058599  0.8113052
## [2,]  0.2258094  1.6606192
## [3,] -0.9775951  1.5463384
## [4,] -0.2629565 -1.2695972
## [5,]  1.0357594 -0.1551543
```

Let's start off by just picking any two of our LASSOs and making predictions with them on this new data. I'll use the same ones where $\lambda = 0.1$ and $\lambda = 0.5$. The resulting output of the code below is predictions of the Y value for each new observation, from each LASSO.

```
predict(fit, newx = nx, s = c(0.1, 0.5))
```

```
##              s1         s2
## [1,] -4.3067990 -2.6135387
## [2,] -4.1244091 -2.3479604
## [3,] -0.1133939  0.4173210
## [4,]  3.3458748  2.4685680
```

```
## [5,] -1.2366422 -0.3380176
```

Use cross-validation to find and use the best LASSO

Above, I just picked any two of the 100 models I estimated to be a predictor. But what does the data tell us is the BEST model to use for prediction? For this, we need cross-validation!

glmnet let's us do this easily with an in-built function cv.glmnet(). How this works is that we take our training data (the objects x and y) and split them into an even set of "folds" (i.e. splits) specified by the "nfolds" parameter in the code below. Then, it will re-train each of the 100 LASSO models 20 times. For each of these 20 training runs, it will use 19/20 splits for training and make predictions on the hold-out 1/20, repeating until each split is held-out once. It computes the Mean Squared Error of these predictions and saves the results of these tests to the cvfit object

Viewing the cvfit object let's us see the Lambda that on-average predicted the best on out-of-sample data. Here, it is where $\lambda = 0.07569$, which is where the average prediction error was smallest. It also shows that in this model, 9 X variables were "important".

```
cvfit <- cv.glmnet(x,
                   y,
                   type.measure = "mse",
                   nfolds = 20)
cvfit
```

```
##
## Call:  cv.glmnet(x = x, y = y, type.measure = "mse", nfolds = 20)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.07569    34   1.021 0.1175       9
## 1se 0.14517    27   1.137 0.1384       8
```

Let's look at the coefficients of the best model:

```
coef(cvfit, s = "lambda.min")
```

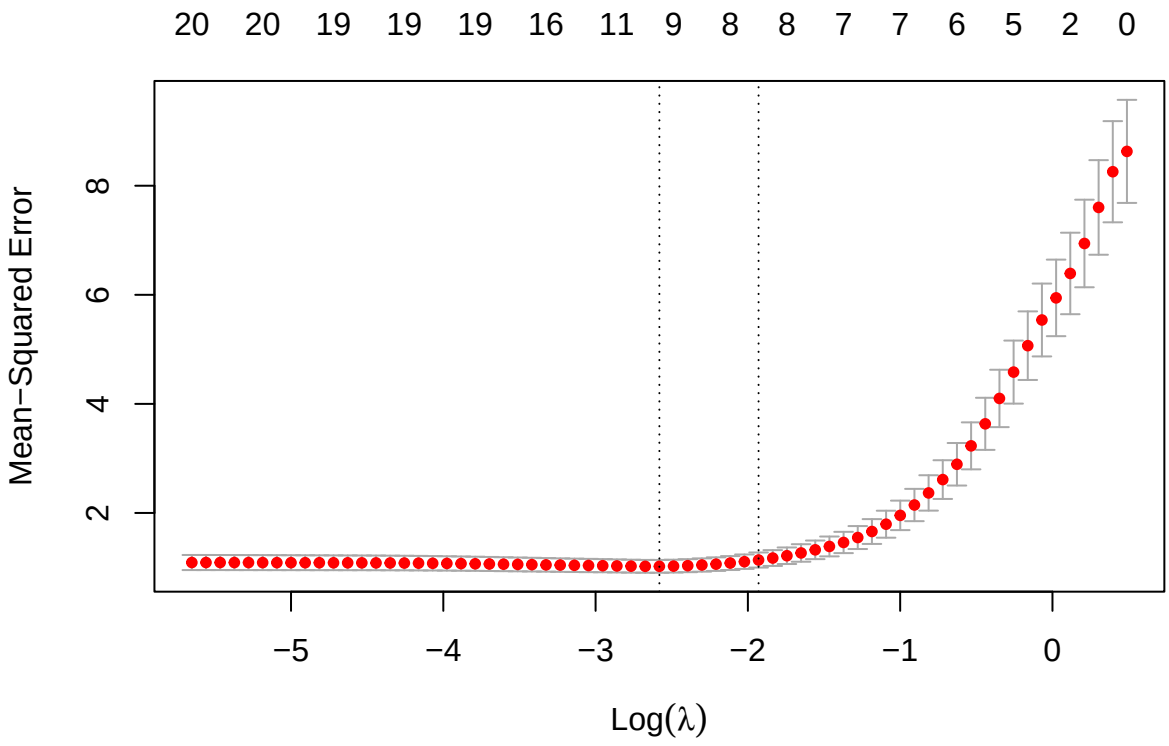
```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.14867414
## V1           1.33377821
## V2           .
## V3           0.69787701
## V4           .
## V5          -0.83726751
## V6           0.54334327
## V7           0.02668633
## V8           0.33741131
## V9           .
## V10          .
## V11          0.17105029
## V12          .
## V13          .
## V14         -1.07552680
## V15          .
## V16          .
## V17          .
```

```
## V18           .
## V19           .
## V20         -1.05278699
```

We can also look at how each value of λ did as far as MSE with the following code. Here's how to interpret this graph:

1. the bottom x-axis is the $\log(\lambda)$ values we tested. In this plot, as we move left to right, λ gets LARGER (the opposite of the last plot), which corresponds to more strict regularization.
2. The y-axis is the MSE of each LASSO model with the upper and lower standard deviations of the squared prediction errors.
3. The top x-axis is the number of non-zero covariates as we increase the regularization intensity
4. The first dotted line is the best LASSO model (lowest MSE), which is where $\log(\lambda) = -2.58$, which is the same as $\lambda = 0.07569$. The second dotted line is the LASSO with the largest lambda (most strict regularization) that produced an MSE within 1 standard deviation of the best LASSO

```
plot(cvfit)
```



Finally, let's make predictions on the new data we generated earlier with the best model:

```
predict(cvfit, newx = nx, s = "lambda.min")
```

```
##      lambda.min
## [1,] -4.4252178
## [2,] -4.2335387
## [3,] -0.1232963
## [4,]  3.4009377
```

```
## [5,] -1.2768042
```