

Lab 10

Y. Samuel Wang

4/18/2022

Lasso and Ridge Regression for linear models

In class, we discussed two ways to fit regressions when the number of covariates is large relative to the number of samples. We'll see how to run the code in R and also explore some properties of the two procedures. We'll also explore how they can be used for GLMs.

First, let's consider linear regression simulate some data. We sample data with 50 covariates and 60 observations. The covariates are correlated with each other. In this first setting, the dependent variable is only a function of variables 1, 5, and 10.

```
set.seed(102)
# If you haven't installed these packages before, run the install.packages commands below
# Package which fits LASSO and Ridge Regression very efficiently
# install.packages("glmnet")
# Package which samples from multivariate Gaussian
# install.packages("mvtnorm")

library(glmnet)
library(mvtnorm)

# Simulate data
n <- 60 # number of samples
p <- 50 # number of covariates
rho <- .3 # correlation between each variable

# Create covariance matrix with 1's on the diagonal and rho on the off-diagonal
sigma <- diag(p)
sigma[abs(row(sigma) - col(sigma)) == 1] <- rho

# Sample the covariates which have the covariance matrix we specified
X <- rmvnorm(n, sigma = sigma)

# the coefficients are 0 everywhere except 1, 5 and 10 where they are .75
b <- rep(0, p)
b[c(1, 5, 10)] <- .75

## Try repeating the exercise, but where lots of coefficients matter
# b[1:30] <- .5

# Create the dependent variable as a function of X1, X5, X10 and some random noise
Y <- X %*% b + rnorm(n)
```

```
# We will also create some test data which we will use later
X.test <- rmvnorm(n * 10, sigma = sigma)
Y.test <- X.test %*% b + rnorm(n * 10)
```

We will use the `glmnet` package to run the Lasso and Ridge regression procedures. Recall that for Lasso, we fix the tuning parameter λ and estimate coefficients that minimize

$$\sum_i (y_i - \hat{y})^2 + \lambda \sum_k |\hat{b}_k|.$$

Using this penalty tends to give solutions \hat{b} that have many zeros. So this can be thought of as a type of model selection. Different values of the tuning parameter λ will give different \hat{b} . The `glmnet` function doesn't require us to specify a single value for λ , but instead can estimate the coefficients for many values different values of λ

```
# x: the covariates
# y: the dependent variable
# alpha: specifies whether to fit Lasso (use 1) or Ridge regression (use 0)
# nlambdas: the number of values to try for lambda, glmnet will automatically pick which specific values
lasso_fit <- glmnet(x = X, y = Y, alpha = 1, nlambdas = 100)
```

Question

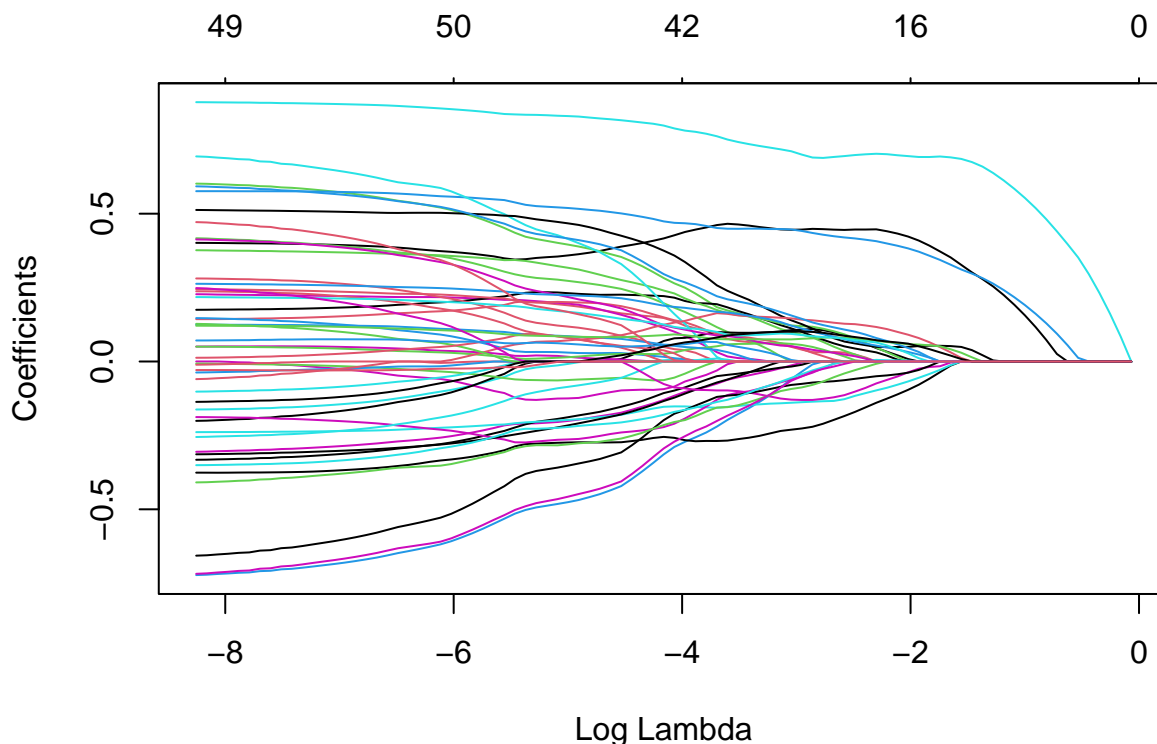
- How do you expect the estimated \hat{b} to change as λ changes? When λ is large, do you expect the estimated coefficients to be larger or smaller? Do you expect there to be more or less zero coefficients as λ increases?
- When would a larger value of λ be better? When would a smaller value of λ be better?
- How would you choose λ ?

The plot below shows the estimated coefficients \hat{b} as we change the value of the penalty parameter λ . The value $\log(\lambda)$ is on the horizontal axis, and the vertical axis shows the value of the estimated coefficients. We can see that as λ increases, many of the coefficients become 0. Does the behavior as λ increases agree with what you thought?

```
# To get the specific values which glmnet chose for lambda
lasso_fit$lambda
```

```
## [1] 0.9346995500 0.8516634181 0.7760039873 0.7070659319 0.6442521432
## [6] 0.5870185585 0.5348694476 0.4873531201 0.4440580121 0.4046091222
## [11] 0.3686647629 0.3359136014 0.3060719629 0.2788813733 0.2541063207
## [16] 0.2315322155 0.2109635316 0.1922221129 0.1751456302 0.1595861751
## [21] 0.1454089792 0.1324912462 0.1207210891 0.1099965602 0.1002247690
## [26] 0.0913210767 0.0832083640 0.0758163623 0.0690810456 0.0629440759
## [31] 0.0573522977 0.0522572776 0.0476148851 0.0433849099 0.0395307141
## [36] 0.0360189144 0.0328190933 0.0299035355 0.0272469879 0.0248264406
## [41] 0.0226209280 0.0206113471 0.0187802918 0.0171119025 0.0155917282
## [46] 0.0142066021 0.0129445267 0.0117945706 0.0107467735 0.0097920597
## [51] 0.0089221601 0.0081295400 0.0074073341 0.0067492869 0.0061496989
## [56] 0.0056033766 0.0051055881 0.0046520217 0.0042387489 0.0038621901
## [61] 0.0035190838 0.0032064581 0.0029216052 0.0026620578 0.0024255680
## [66] 0.0022100872 0.0020137491 0.0018348532 0.0016718499 0.0015233274
## [71] 0.0013879992 0.0012646931 0.0011523413 0.0010499705 0.0009566940
## [76] 0.0008717039 0.0007942642 0.0007237039 0.0006594121 0.0006008317
## [81] 0.0005474555 0.0004988210 0.0004545072 0.0004141300 0.0003773398
## [86] 0.0003438180 0.0003132741 0.0002854437 0.0002600857
```

```
# specifying "lambda" tells R to put log(lambda) on the horizontal axis
plot(lasso_fit, "lambda")
```



We can see what the estimated coefficients were at specific choices of λ . In particular $\log(.05) \approx -3$ and $\log(.2) \approx -1.6$. As we can see in the printouts (and the plot above), when $\lambda = .05$ there are many non-zero

coefficients, but some coefficients are estimated to be 0. But when $\lambda = .2$, we get many more zero coefficients, and only a few coefficients are non-zero.

```
# we can get the coefficients at a specific choice of lambda  
coef(lasso_fit, s = .05)
```

```
## 51 x 1 sparse Matrix of class "dgCMatrix"  
##              s1  
## (Intercept)  0.173040909  
## V1          0.445886743  
## V2          0.044657101  
## V3          0.095582093  
## V4         -0.046472353  
## V5          0.705333267  
## V6          .  
## V7          0.105920455  
## V8          .  
## V9          .  
## V10         0.444042964  
## V11         .  
## V12         .  
## V13         0.106781173  
## V14         0.046724553  
## V15         .  
## V16         .  
## V17         .  
## V18         .  
## V19        -0.228536283  
## V20         .  
## V21         0.120688862  
## V22         .  
## V23         .  
## V24        -0.008342799  
## V25        -0.014029382  
## V26         .  
## V27         0.057640260  
## V28         0.002491267  
## V29        -0.138551344  
## V30        -0.055099599  
## V31         0.132588589  
## V32         0.143610113  
## V33         .  
## V34         0.149809355  
## V35         .  
## V36         0.048852635  
## V37         .  
## V38         .  
## V39         0.075752743  
## V40         0.116253054  
## V41         0.091309573  
## V42        -0.129567477  
## V43        -0.077386440  
## V44         .  
## V45        -0.090150158  
## V46         .
```

```
## V47      -0.048884078
## V48      -0.057507470
## V49       0.100932589
## V50       .
```

```
coef(lasso_fit, s = .2)
```

```
## 51 x 1 sparse Matrix of class "dgCMatrix"
```

```
##              s1
## (Intercept) 0.190904039
## V1          0.341526310
## V2          .
## V3          .
## V4          .
## V5          0.688312339
## V6          .
## V7          .
## V8          .
## V9          .
## V10         0.320715640
## V11         .
## V12         .
## V13         .
## V14         .
## V15         .
## V16         .
## V17         .
## V18         .
## V19        -0.008114198
## V20         .
## V21         .
## V22         .
## V23         .
## V24         .
## V25         .
## V26         .
## V27         .
## V28         .
## V29        -0.007161578
## V30         .
## V31         0.050953310
## V32         0.023084081
## V33         .
## V34         .
## V35         .
## V36         .
## V37         .
## V38         .
## V39         0.036493776
## V40         .
## V41         .
## V42        -0.002019722
## V43         .
## V44         .
## V45         .
```

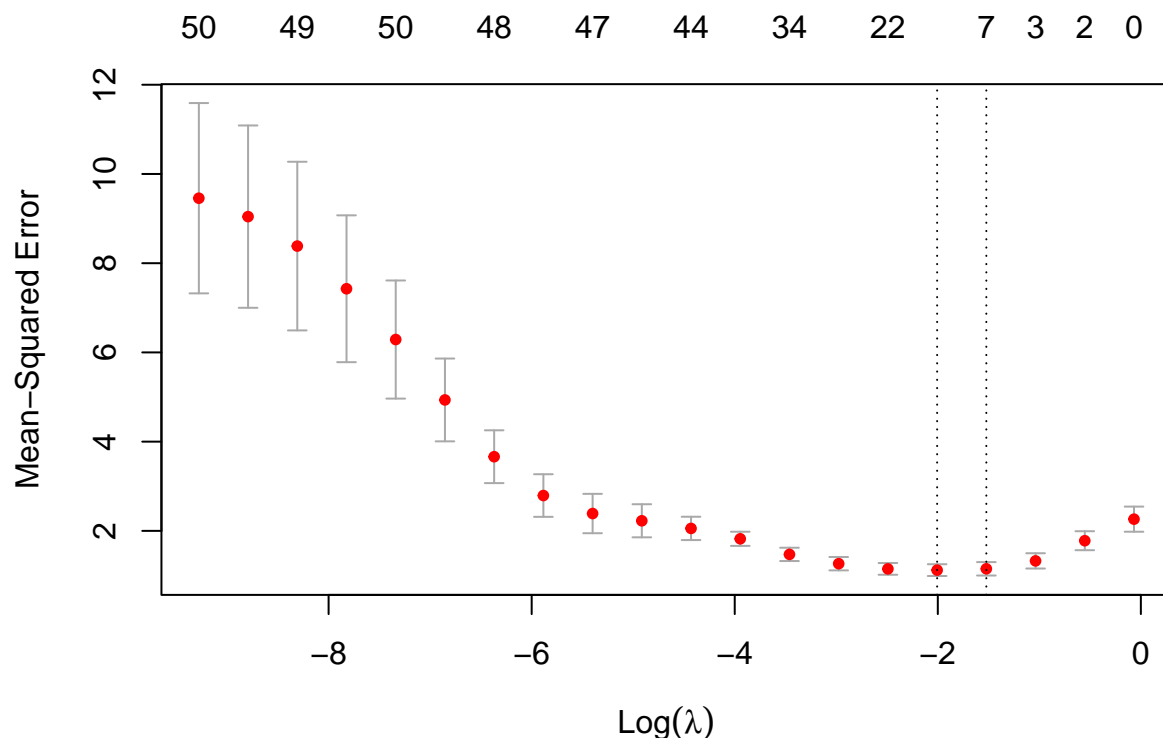
```
## V46      .
## V47      .
## V48      .
## V49      0.015094339
## V50      .
```

We can use cross validation to choose an appropriate λ ! You've seen this function in a previous lab when doing cross validation for linear regression without the Lasso penalty. In this case, we do 10 fold validation. Recall that this means we split the data into 10 equal subsets. First, pick a specific value for λ . For each subset, we hold it out and fit a lasso regression with the λ value to the remaining data. We then measure our predictive accuracy on the subset we held out of our data fitting. We can see the λ value which has the best cross validation error. For reference, we also look at the mean squared prediction error on the new test data which was not used to fit the model

```
# Mostly the same input as before, but use the function cv.glmnet
# nfolds: specifies how many folds to use for K fold validation
cv_lasso_fit <- cv.glmnet(x = X, y = Y, alpha = 1, nlambda = 20, nfolds = 10)
cv_lasso_fit$lambda.min
```

```
## [1] 0.1344518
```

```
plot(cv_lasso_fit)
```



```
# The two vertical lines on the plot above are:
# The value of lambda which has the lowest CV error
cv_lasso_fit$lambda.min
```

```
## [1] 0.1344518
```

```
# the largest value of lambda which has a CV error within 1 se of the lowest CV error
cv_lasso_fit$lambda.1se
```

```
## [1] 0.2183198
```

```
## use lasso_fit$lambda.min to get the lambda value with the best cv error
coef(cv_lasso_fit, s = cv_lasso_fit$lambda.min)
```

```
## 51 x 1 sparse Matrix of class "dgCMatrix"
```

```
##              s1
## (Intercept) 0.153874696
## V1          0.420532586
## V2          .
## V3          .
## V4          .
## V5          0.694126508
## V6          .
## V7          .
## V8          .
## V9          .
## V10         0.380976300
## V11         .
## V12         .
## V13         0.006580361
## V14         .
## V15         .
## V16         .
## V17         .
## V18         .
## V19        -0.093200694
## V20         .
## V21         .
## V22         .
## V23         .
## V24         .
## V25         .
## V26         .
## V27         .
## V28         .
## V29        -0.065602672
## V30         .
## V31         0.049975128
## V32         0.088850090
## V33         .
## V34         0.043810082
## V35         0.002353584
## V36         .
## V37         .
## V38         .
## V39         0.058234302
## V40         0.030940605
## V41         0.025151747
## V42        -0.034521702
## V43        -0.036501589
## V44         .
## V45         .
## V46         .
## V47         .
## V48         .
```

```
## V49          0.059017038
## V50          .
```

```
coef(cv_lasso_fit, s = cv_lasso_fit$lambda.1se)
```

```
## 51 x 1 sparse Matrix of class "dgCMatrix"
```

```
##              s1
## (Intercept) 0.198229005
## V1          0.316760437
## V2          .
## V3          .
## V4          .
## V5          0.682035705
## V6          .
## V7          .
## V8          .
## V9          .
## V10         0.306236335
## V11         .
## V12         .
## V13         .
## V14         .
## V15         .
## V16         .
## V17         .
## V18         .
## V19         .
## V20         .
## V21         .
## V22         .
## V23         .
## V24         .
## V25         .
## V26         .
## V27         .
## V28         .
## V29         .
## V30         .
## V31         0.046672706
## V32         0.004964859
## V33         .
## V34         .
## V35         .
## V36         .
## V37         .
## V38         .
## V39         0.024421275
## V40         .
## V41         .
## V42         .
## V43         .
## V44         .
## V45         .
## V46         .
## V47         .
```



```

## V48      .
## V49      0.002546194
## V50      .

# the predict function takes the model we fit 'cv_lasso_fit'
# using the 's' argument, we specify which value of lambda we want to use
# we could use either the lambda with the smallest CV error or the largest lambda
# with CV error within 1 se of the lowest CV error
# newx is the covariates for which we are making predictions. In this case, it's the test data
lasso_predicted <- predict(cv_lasso_fit, s = cv_lasso_fit$lambda.min, newx = X.test, type = "response")
# lasso_predicted <- predict(cv_lasso_fit, s = cv_lasso_fit$lambda.1se, newx = X.test)

coef(cv_lasso_fit, s = cv_lasso_fit$lambda.min)

## 51 x 1 sparse Matrix of class "dgCMatrix"
##              s1
## (Intercept)  0.153874696
## V1          0.420532586
## V2          .
## V3          .
## V4          .
## V5          0.694126508
## V6          .
## V7          .
## V8          .
## V9          .
## V10         0.380976300
## V11         .
## V12         .
## V13         0.006580361
## V14         .
## V15         .
## V16         .
## V17         .
## V18         .
## V19        -0.093200694
## V20         .
## V21         .
## V22         .
## V23         .
## V24         .
## V25         .
## V26         .
## V27         .
## V28         .
## V29        -0.065602672
## V30         .
## V31         0.049975128
## V32         0.088850090
## V33         .
## V34         0.043810082
## V35         0.002353584
## V36         .
## V37         .

```

```
## V38      .
## V39      0.058234302
## V40      0.030940605
## V41      0.025151747
## V42     -0.034521702
## V43     -0.036501589
## V44      .
## V45      .
## V46      .
## V47      .
## V48      .
## V49      0.059017038
## V50      .
```

```
# We can calculate the mean squared prediction error on test data
lasso_test_error <- mean((Y.test - lasso_predicted)^2)
lasso_test_error
```

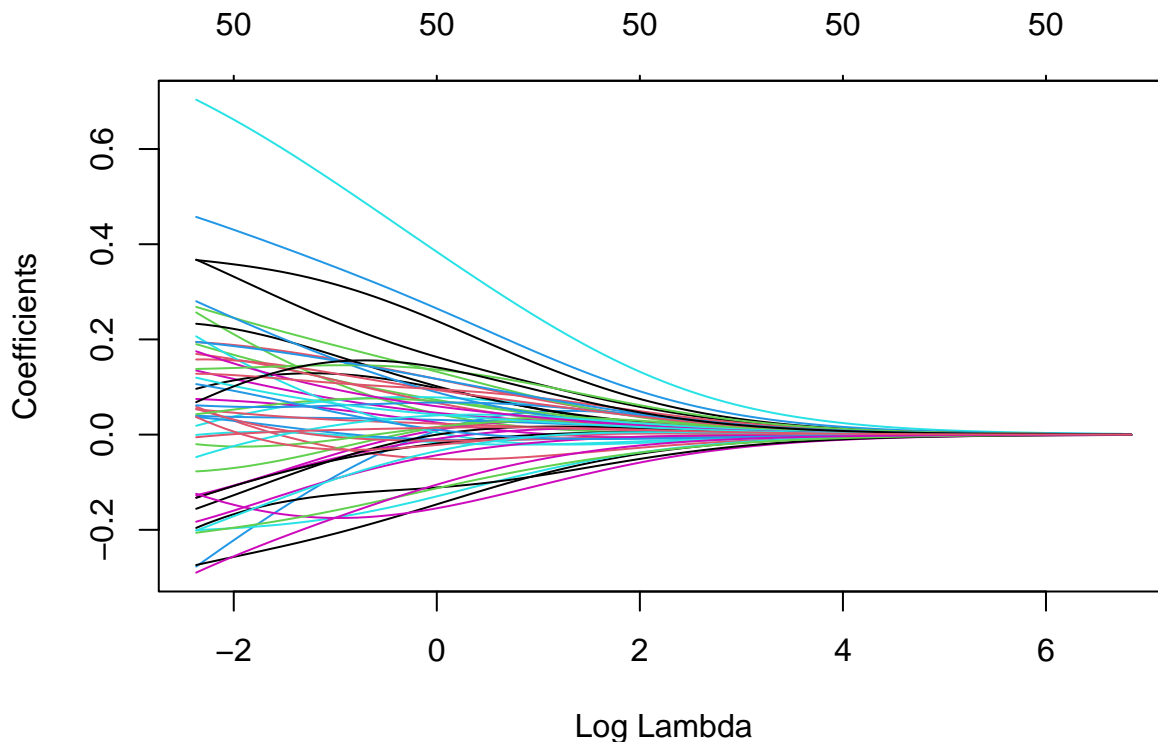
```
## [1] 1.296928
```

We can now try the same thing, but with ridge regression instead. Recall that ridge regression solve the following problem

$$\sum_i (y_i - \hat{y})^2 + \lambda \sum_k \hat{b}_k^2.$$

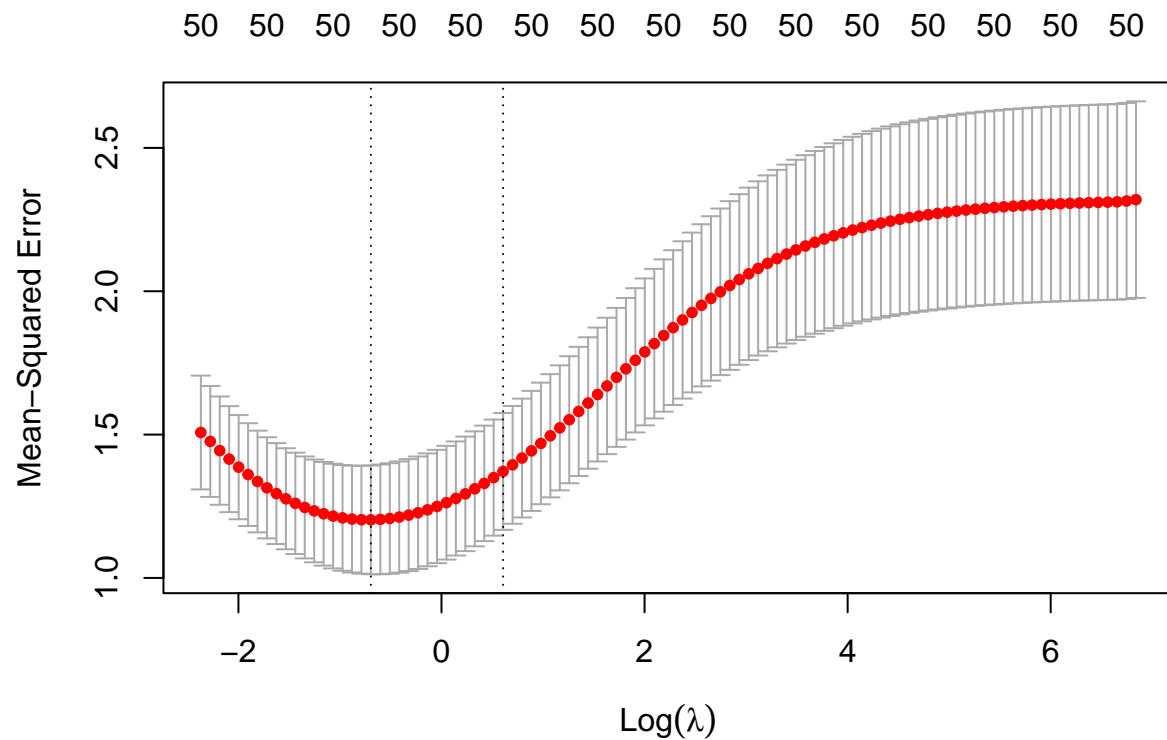
Everything is the same, except we change `alpha = 0`. As opposed to above, we now see that the as λ increases, the coefficients get smaller but do not go to 0 exactly (though they look very close in the plot). Thus, ridge regression shrinks the values towards 0, but does not select specific covariates to include/exclude. This is one of the major differences between lasso and ridge. Ridge doesn't do model selection (i.e., include/exclude variable) so the fitted model can sometimes be a bit harder to interpret.

```
# Change alpha = 0 to do ridge
ridge_fit <- glmnet(x = X, y = Y, alpha = 0, nlambda = 100)
plot(ridge_fit, "lambda")
```



We can also do cross validation to select a value of λ and look at the prediction error for ridge regression.

```
# Mostly the same input as before
# Change alpha = 0 to do ridge
cv_ridge_fit <- cv.glmnet(x = X, y = Y, alpha = 0, nlambda = 100, nfolds = 10)
plot(cv_ridge_fit)
```



```
cv_ridge_fit$lambda.min
```

```
## [1] 0.498821
```

```
cv_ridge_fit$lambda.1se
```

```
## [1] 1.834853
```

```
## use lasso_fit$lambda.min to get the lambda value with the best cv error
```

```
coef(cv_ridge_fit, s = cv_ridge_fit$lambda.min)
```

```
## 51 x 1 sparse Matrix of class "dgCMatrix"
```

```
##              s1
## (Intercept) 0.230778955
## V1          0.295682919
## V2          0.146295308
## V3          0.084876091
## V4         -0.053280181
## V5          0.485462374
## V6          0.064416591
## V7          0.145739143
## V8         -0.038289069
## V9          0.100879022
## V10         0.326508683
## V11         0.076820314
## V12        -0.022678097
## V13         0.122247934
## V14         0.098934925
## V15        -0.014556417
## V16         0.033907375
## V17        -0.005559822
## V18         0.045221595
```

```
## V19      -0.190330056
## V20      0.028354465
## V21      0.170639634
## V22     -0.002321081
## V23      0.042541340
## V24     -0.035072356
## V25     -0.040874837
## V26     -0.009311421
## V27      0.075940096
## V28      0.063476905
## V29     -0.162973631
## V30     -0.075115057
## V31      0.211817823
## V32      0.117369772
## V33     -0.002472361
## V34      0.134879960
## V35      0.028234708
## V36      0.082457253
## V37     -0.032628190
## V38      0.103965445
## V39      0.145315943
## V40      0.144300417
## V41      0.057033557
## V42     -0.173438773
## V43     -0.120713192
## V44     -0.031598499
## V45     -0.145558107
## V46      0.034259785
## V47     -0.072687211
## V48     -0.152239181
## V49      0.155948774
## V50      0.011426067
```

```
# the predict function takes the model we fit 'cv_lasso_fit'
# using the 's' argument, we specify which value of lambda we want to use
# we could use either the lambda with the smallest CV error or the largest lambda
# with CV error within 1 se of the lowest CV error
# newx is the covariates for which we are making predictions. In this case, it's the test data
ridge_predicted <- predict(cv_lasso_fit, s = cv_lasso_fit$lambda.min, newx = X.test)
# ridge_predicted <- predict(cv_lasso_fit, s = cv_lasso_fit$lambda.1se, newx = X.test)

# We can calculate the mean squared prediction error on test data
ridge_test_error <- mean((Y.test - ridge_predicted)^2)
ridge_test_error
```

```
## [1] 2.015069
```

We can compare the predictive accuracy for this data set and see that the Lasso does better in predictive accuracy. It also identifies a model pretty close to the true model, while ridge doesn't select any specific variables. For comparison, we the predictive accuracy an ordinary linear model (i.e., unpenalized regression).

```
# Error for regression with no penalty
# when p > n, we can't actually compute this

mean((Y.test - predict(lm(Y~X), newx = X.test))^2)
```

```
## [1] 5.237853  
# Lasso error  
lasso_test_error
```

```
## [1] 1.296928  
# Ridge Error  
ridge_test_error
```

```
## [1] 2.015069
```

Questions

- It looks like Lasso does better than the ridge here, but both do significantly better than unpenalized linear regression. Are there situations where you think ridge might do better than Lasso? Try repeating the experiment, but in the true model, let lots of coefficients be non-zero. What works best in that setting?

Lasso and Ridge Regression for GLMs

Coming back to GLMs, we can also fit GLMs with the Lasso and Ridge penalties. Instead of trying to minimize the RSS + penalty, we instead try to maximize the log-likelihood - penalty:

$$\ell(\theta; Y) - \lambda \sum_k |\hat{b}_k|$$

or

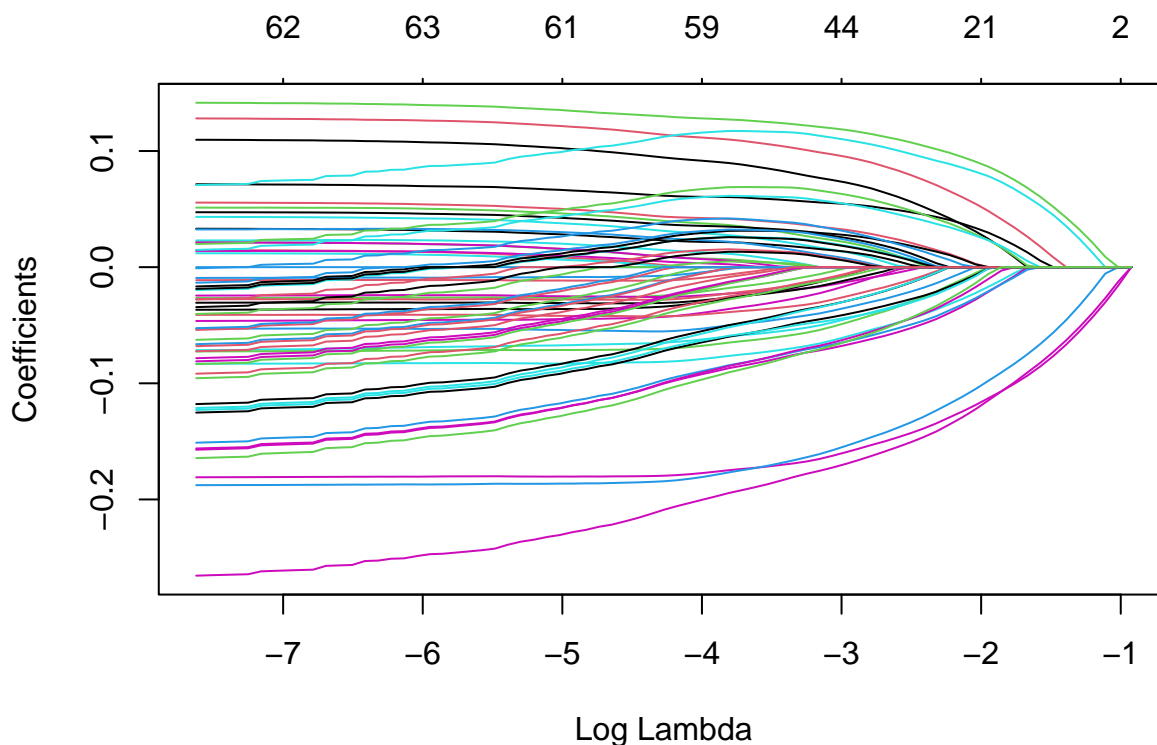
$$\ell(\theta; Y) - \lambda \sum_k \hat{b}_k^2.$$

In terms of computation, things stay very similar. We can try it out with the Poisson regression we used in lab last week that considered penalties in football games. This time, as covariates we will consider the specific home team and away team for the game. Thus, we might see if games have less penalties when certain teams are playing. This means we have roughly 60 covariates (which are dummy variables). So the number of covariates in this case is quite large. Examining the estimated coefficients, we see that there are some teams which are estimated to have a positive/negative association with the number of penalties.

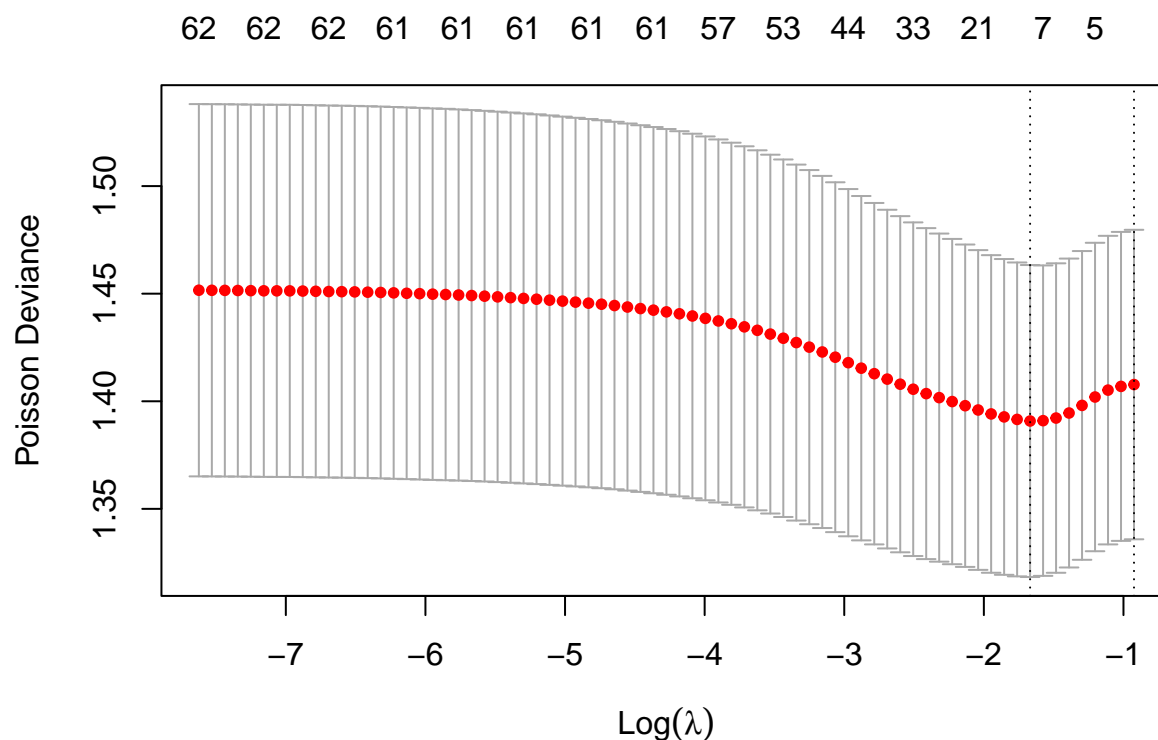
```
library(readr)
# Read in the data from last week
penalty_data <- read_csv("https://raw.githubusercontent.com/ysamwang/btry6020_sp22/main/lectureData/penalty_data.csv")

## glmnet does not automatically create dummy variables like when you use lm or glm
## but we can use the model.matrix function to create the matrix with dummy variables
team_dummies <- model.matrix(~ . -1, data = data.frame(penalty_data$home_team, penalty_data$away_team))

# Same as before, but now we specify "family = poisson"
lasso_poisson <- glmnet(y = penalty_data$penalty_count, x = team_dummies, family = "poisson", alpha = 1)
plot(lasso_poisson, "lambda")
```



```
# now using cv to pick a lambda
cv_lasso_poisson <- cv.glmnet(y = penalty_data$penalty_count, x = team_dummies, family = "poisson", alpha = 1)
plot(cv_lasso_poisson)
```



```
coef(cv_lasso_poisson, s = cv_lasso_poisson$lambda.min)
```

```
## 64 x 1 sparse Matrix of class "dgCMatrix"
##                               s1
## (Intercept)                2.5179016983
## penalty_data.home_teamARI .
## penalty_data.home_teamATL .
## penalty_data.home_teamBAL .
## penalty_data.home_teamBUF .
## penalty_data.home_teamCAR .
## penalty_data.home_teamCHI .
## penalty_data.home_teamCIN .
## penalty_data.home_teamCLE .
## penalty_data.home_teamDAL .
## penalty_data.home_teamDEN .
## penalty_data.home_teamDET .
## penalty_data.home_teamGB -0.0934660302
## penalty_data.home_teamHOU .
## penalty_data.home_teamIND .
## penalty_data.home_teamJAX .
## penalty_data.home_teamKC .
## penalty_data.home_teamLA .
## penalty_data.home_teamLAC .
## penalty_data.home_teamLV  0.0127329963
## penalty_data.home_teamMIA .
## penalty_data.home_teamMIN .
## penalty_data.home_teamNE -0.0750256973
```



```

## penalty_data.home_teamNO .
## penalty_data.home_teamNYG .
## penalty_data.home_teamNYJ .
## penalty_data.home_teamPHI 0.0262235837
## penalty_data.home_teamPIT 0.0682957558
## penalty_data.home_teamSEA .
## penalty_data.home_teamSF .
## penalty_data.home_teamTB .
## penalty_data.home_teamTEN .
## penalty_data.home_teamWAS .
## penalty_data.away_teamATL 0.0013666686
## penalty_data.away_teamBAL .
## penalty_data.away_teamBUF .
## penalty_data.away_teamCAR -0.0002690785
## penalty_data.away_teamCHI .
## penalty_data.away_teamCIN .
## penalty_data.away_teamCLE .
## penalty_data.away_teamDAL .
## penalty_data.away_teamDEN 0.0052426273
## penalty_data.away_teamDET .
## penalty_data.away_teamGB .
## penalty_data.away_teamHOU .
## penalty_data.away_teamIND .
## penalty_data.away_teamJAX .
## penalty_data.away_teamKC 0.0604364703
## penalty_data.away_teamLA -0.0907226577
## penalty_data.away_teamLAC .
## penalty_data.away_teamLV .
## penalty_data.away_teamMIA .
## penalty_data.away_teamMIN -0.0021320077
## penalty_data.away_teamNE .
## penalty_data.away_teamNO .
## penalty_data.away_teamNYG .
## penalty_data.away_teamNYJ .
## penalty_data.away_teamPHI .
## penalty_data.away_teamPIT .
## penalty_data.away_teamSEA .
## penalty_data.away_teamSF .
## penalty_data.away_teamTB .
## penalty_data.away_teamTEN .
## penalty_data.away_teamWAS .

```

Discussion Questions

- How do you expect the estimated \hat{b} to change as λ changes? When λ is large, do you expect the estimated coefficients to be larger or smaller? Do you expect there to be more or less zero coefficients as λ increases?
 - When λ is larger, the penalty should push the solution towards smaller values of \hat{b} . Thus, we would expect the estimated coefficients to be smaller. In addition, we would expect there to be more coefficients which are set to zero.
- When would a larger value of λ be better? When would a smaller value of λ be better?
- When n is very large relative to p , then just using unpenalized regression should do pretty well. Thus, we should expect using a smaller value of λ should also work okay. When n is small relative to p , we expect unpenalized regression should do poorly and shrinking the regression coefficients towards 0 should be more helpful. Thus, a larger value of λ is probably more helpful.
- How would you choose λ ?
 - We use cross-validation!
- It looks like Lasso does better than the ridge here, but both do significantly better than unpenalized linear regression. Are there situations where you think ridge might do better than Lasso? Try repeating the experiment, but in the true model, let lots of coefficients be non-zero. What works best in that setting?
 - In this simulation, the true model is one where most coefficients are 0. Thus lasso does well in this scenario. However, if the true model includes many covariates with non-zero coefficients, we would expect ridge regression to do better because it still includes all the variables.