Stat 432 Homework 4

Giang Le (gianghl2)

Assigned: Sep 13, 2021; Due: 11:59 PM CT, Sep 21, 2021

Contents

Question 1: The Bias-Variance Trade-Off Simulation

In the code block below, I generated a sequence of 100 lambda values from 0 to 0.5, and a matrix of (1000, 100) dimension to store beta_ones from 1000 simulations (stored in the rows) and 100 lambda values (stored in the columns). I looped through the lambda values in the outer loop and the simulations in the inner loop, inside which I generated a X matrix for two highly correlated variables (cor=0.9)) and I randomly generated a y vector. β_1 is calculated and stored in the matrix by the ridge formula.

```
library(MASS)

set.seed(662095561)
 # number of researchers
 nsim = 1000
 # number of observations
 n = 100

 # lambdas
 lambdas <- seq(0, 0.5, by = 0.5/(100-1))

 # set up the matrix of ridge betas
 betaones  <- matrix(NA, nsim, length(lambdas))

 for (i in 1:length(lambdas)) {
   for (j in 1:nsim)
    {
   # create highly correlated variables and a linear model
   X = mvrnorm(n, c(0, 0), matrix(c(1,0.9, 0.9, 1), 2,2))
   y = rnorm(n, mean = X[,1] + X[,2])

   betaones[j, i] <- (solve(t(X) %*% X + lambdas[i]*n*diag(2)) %*% t(X) %*% y)[1]
   }
 }
```

Upon obtaining a matrix of $\hat{\beta}_1$, I calculated the bias (the difference between mean $\hat{\beta}_1$ and 1 the truth value) and the variance. I also calculated the sum of bias_squared and variance.

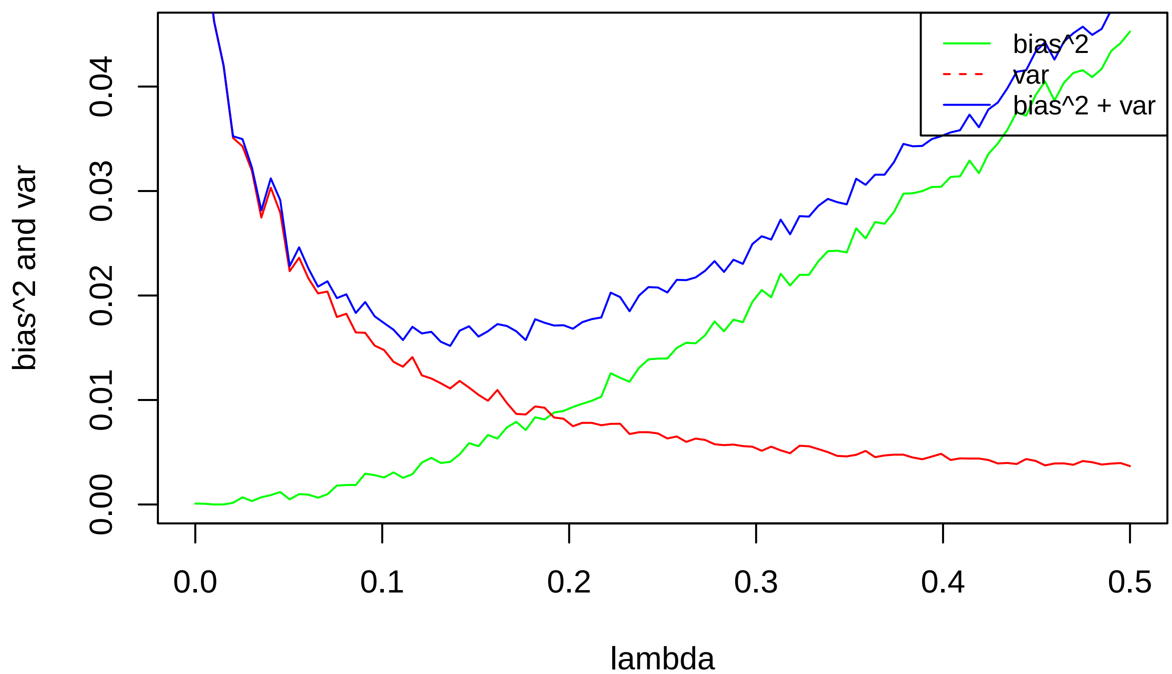
```
variance <- apply(betaones, 2, var)
bias <- colMeans(betaones) - 1
bias_squared <- bias^2
```

```
bias_variance <- bias_squared + variance
```

- Lastly, what have you observed in terms of the trend for Bias^2, Variance, and their sum, respectively? What is causing these? And if you are asked to choose a λ value that works the best, which value would you choose?

In this plot, I observed that the bias^2 increases with increasing lambdas while the variance decreases with increasing lambdas. The two quantities therefore are negatively correlated. The sum has a convex U-shape. These relationships demonstrate the bias-variance tradeoff, where a model with low bias has a higher variance and vice versa. This is because the diagonal matrix makes the solution more stable, reduces the variance of the estimator (the eigenvalues of XTX are large) but at the same time bias increases. I will choose lambda that minimizes the bias^2 + variance, which is approximately 0.1 in this case.

```
# Plot three quantities against lambdas.
plot(x=lambdas,y=bias_squared, type="l", col="green", xlab="lambda",ylab="bias^2 and var")
lines(x=lambdas,y=variance,col="red")
lines(x=lambdas,y=bias_variance,col="blue")
legend("topright", legend=c("bias^2", "var", "bias^2 + var"),
       col=c("green", "red", "blue"), lty=1:2, cex=0.8)
```



Question 2: The Cross-Validation

We used the `mtcars` data in the lecture notes, and also introduced the k-fold cross-validation. For this question you need to complete the following with the `mtcars` data:

- Write a 5-fold cross-validation code by yourself, using the `lm.ridge()` function to fit the model and predict on the testing data. Choose an appropriate range of lambda values based on how this function specifies the penalty. Obtain the cross-validation error corresponding to each λ and produce an intuitive plot to how it changes over different λ. What is the best penalty level you obtained from this procedure? Compare that with the GCV result. Please note that you should clearly state the intention of each step of your code and state your result.
- Use the `cv.glmnet()` function from the `glmnet` package to perform a 5-fold cross-validation using their built-in feature. Produce the cross-validation error plot against λ values. Report the `lambda.min` and `lambda.1se` selected λ value.

First I inspect the mtcars dataset. This dataset has 32 observations and 11 variables.

```
dim(mtcars)
```

```
## [1] 32 11
```

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

I create 5 folds (6 to 8 observations in the test fold, the rest would be the training data). I do this by selecting 6 observations to use as the test fold for simplicity (in the first fold, there are 8 observations in the test fold). The train data would be the rest.

```
set.seed(662095561)
test_folds <- split(sample(mtcars), rep(1:5, each=6))
```

```
## Warning in split.default(x = seq_len(nrow(x)), f = f, drop = drop, ...): data
## length is not a multiple of split variable
```

```
test1 <- as.data.frame(test_folds[1])
train1 <- mtcars[!row.names(mtcars) %in% row.names(test1),]
test2 <- as.data.frame(test_folds[2])
train2 <- mtcars[!row.names(mtcars) %in% row.names(test2),]
test3 <- as.data.frame(test_folds[3])
train3 <- mtcars[!row.names(mtcars) %in% row.names(test3),]
test4 <- as.data.frame(test_folds[4])
train4 <- mtcars[!row.names(mtcars) %in% row.names(test4),]
test5 <- as.data.frame(test_folds[5])
train5 <- mtcars[!row.names(mtcars) %in% row.names(test5),]
```

Using the 5 folds created, I fit 5 lm.ridge models below:

```
library(MASS)
fit1 = lm.ridge(mpg ~., data = train1, lambda = seq(0, 40, by=1))
fit2 = lm.ridge(mpg ~., data = train2, lambda = seq(0, 40, by=1))
fit3 = lm.ridge(mpg ~., data = train3, lambda = seq(0, 40, by=1))
fit4 = lm.ridge(mpg ~., data = train4, lambda = seq(0, 40, by=1))
```

```
fit5 = lm.ridge(mpg ~., data = train5, lambda = seq(0, 40, by=1))
```

Now I use the test data to predict values for y for model 1 and other models. I do this by extracting only the predictors from the test data and created a matrix for the predicted y values by observation x lambda value. After that I computed the MSE of each cross validation model for each lambda. I also plotted the MSE values vs. lambda. Towards the end, I averaged the MSE across models to choose the optimal penalty level for our dataset.

```
test1_predictors <- subset(as.matrix(cbind(const=1, test1)), select = -X1.mpg)
test1_predictors
```

```
##                   const X1.disp X1.carb X1.gear X1.wt X1.drat X1.hp X1.qsec
## Mazda RX4             1     160       4       4 2.620    3.90   110   16.46
## Mazda RX4 Wag         1     160       4       4 2.875    3.90   110   17.02
## Datsun 710            1     108       1       4 2.320    3.85    93   18.61
## Hornet 4 Drive        1     258       1       3 3.215    3.08   110   19.44
## Hornet Sportabout     1     360       2       3 3.440    3.15   175   17.02
## Valiant               1     225       1       3 3.460    2.76   105   20.22
## Maserati Bora         1     301       8       5 3.570    3.54   335   14.60
## Volvo 142E            1     121       2       4 2.780    4.11   109   18.60
##                   X1.cyl X1.vs X1.am
## Mazda RX4              6     0     1
## Mazda RX4 Wag          6     0     1
## Datsun 710             4     1     1
## Hornet 4 Drive         6     1     0
## Hornet Sportabout      8     0     0
## Valiant                6     1     0
## Maserati Bora          8     0     1
## Volvo 142E             4     1     1
```

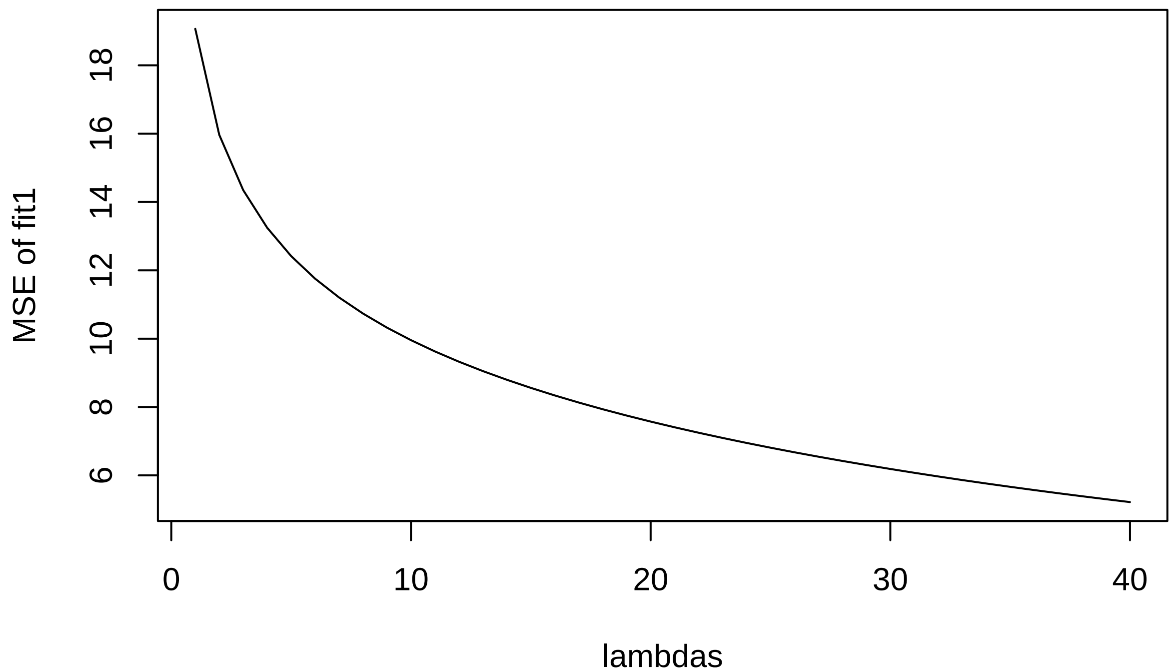
```
coef(fit1)[1,]
```

```
##                      cyl         disp           hp         drat           wt
## 30.746329567 -0.458514055 -0.007861804 -0.013547677 -0.385616593 -0.411576245
##         qsec           vs           am         gear         carb
##  0.042857616  2.248256418  6.164909808 -0.542274562 -1.036887110
```

```
y.pred1 <- matrix(NA, 8, 40)

# Because I cannot guarantee the order of predictors.
for (i in 1:40) {
#   y.pred1[,i] <- test1_predictors[, c("X1.cyl")] * coef(fit1)[i,c("cyl")]
   y.pred1[,i] <- test1_predictors[, c("const")] * coef(fit1)[i,1] +
      test1_predictors[, c("X1.disp")] * coef(fit1)[i,c("disp")] +
      test1_predictors[, c("X1.carb")] * coef(fit1)[i,c("carb")] +
      test1_predictors[, c("X1.gear")] * coef(fit1)[i,c("gear")] +
      test1_predictors[, c("X1.wt")] * coef(fit1)[i,c("wt")] +
      test1_predictors[, c("X1.drat")] * coef(fit1)[i,c("drat")] +
      test1_predictors[, c("X1.hp")] * coef(fit1)[i,c("hp")] +
      test1_predictors[, c("X1.qsec")] * coef(fit1)[i,c("qsec")] +
      test1_predictors[, c("X1.cyl")] * coef(fit1)[i,c("cyl")] +
      test1_predictors[, c("X1.vs")] * coef(fit1)[i,c("vs")] +
      test1_predictors[, c("X1.am")] * coef(fit1)[i,c("am")]
}
errors1 <- (y.pred1 - test1[,c("X1.mpg")])^2
colMeans(errors1) #MSE
```

```
##  [1] 19.067697 15.967495 14.347677 13.246417 12.418066 11.756300 11.205893
##  [8] 10.734786 10.322854  9.956700  9.626986  9.326959  9.051585  8.797019
## [15]  8.560260  8.338921  8.131075  7.935146  7.749827  7.574023  7.406809
## [22]  7.247393  7.095097  6.949332  6.809583  6.675401  6.546386  6.422186
## [29]  6.302486  6.187003  6.075483  5.967696  5.863434  5.762508  5.664743
## [36]  5.569982  5.478079  5.388898  5.302315  5.218215
```

```
plot(colMeans(errors1), type="l", xlab="lambdas", ylab="MSE of fit1")
```



Now I use the test data to predict values for y for model 2

```
test2_predictors <- subset(as.matrix(cbind(const=1, test2)), select = -X2.mpg)
coef(fit2)[1,]
```

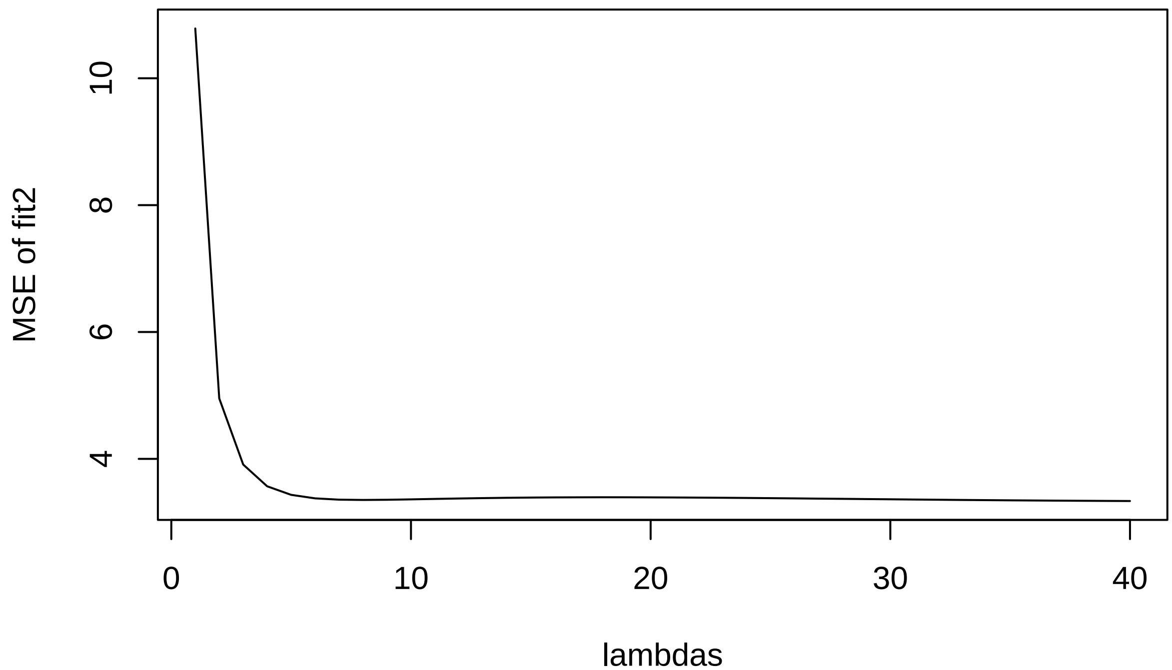
```
##                        cyl         disp           hp         drat           wt
## -15.68912541  -0.23736551   0.02948863  -0.01334320   1.27566875  -5.93578829
##         qsec           vs           am         gear         carb
##   2.29273128  -1.51196255   2.04458360   1.35104038   0.31993580
```

```
y.pred2 <- matrix(NA, 6, 40)

# Because I cannot guarantee the order of predictors.
for (i in 1:40) {
   y.pred2[,i] <- test2_predictors[, c("const")] * coef(fit2)[i,1] +
      test2_predictors[, c("X2.disp")] * coef(fit2)[i,c("disp")] +
      test2_predictors[, c("X2.carb")] * coef(fit2)[i,c("carb")] +
      test2_predictors[, c("X2.gear")] * coef(fit2)[i,c("gear")] +
      test2_predictors[, c("X2.wt")] * coef(fit2)[i,c("wt")] +
      test2_predictors[, c("X2.drat")] * coef(fit2)[i,c("drat")] +
      test2_predictors[, c("X2.hp")] * coef(fit2)[i,c("hp")] +
      test2_predictors[, c("X2.qsec")] * coef(fit2)[i,c("qsec")] +
      test2_predictors[, c("X2.cyl")] * coef(fit2)[i,c("cyl")] +
      test2_predictors[, c("X2.vs")] * coef(fit2)[i,c("vs")] +
```

```
    test2_predictors[, c("X2.am")] * coef(fit2)[i,c("am")]
}

errors2 <- (y.pred2 - test2[,c("X2.mpg")])^2
colMeans(errors2) #MSE
```

```
##  [1] 10.784905  4.953794  3.910479  3.567988  3.432879  3.377424  3.356850
##  [8]  3.352439  3.355479  3.361636  3.368684  3.375473  3.381433  3.386309
## [15]  3.390022  3.392596  3.394103  3.394644  3.394331  3.393278  3.391596
## [22]  3.389388  3.386752  3.383776  3.380541  3.377120  3.373577  3.369971
## [29]  3.366354  3.362772  3.359264  3.355868  3.352615  3.349531  3.346642
## [36]  3.343968  3.341527  3.339334  3.337403  3.335745
```

```
plot(colMeans(errors2), type="l", xlab="lambdas",ylab="MSE of fit2")
```



Now I use the test data to predict values for y for model 3

```
test3_predictors <- subset(as.matrix(cbind(const=1, test3)), select = -X3.mpg)
coef(fit3)[1,]
```

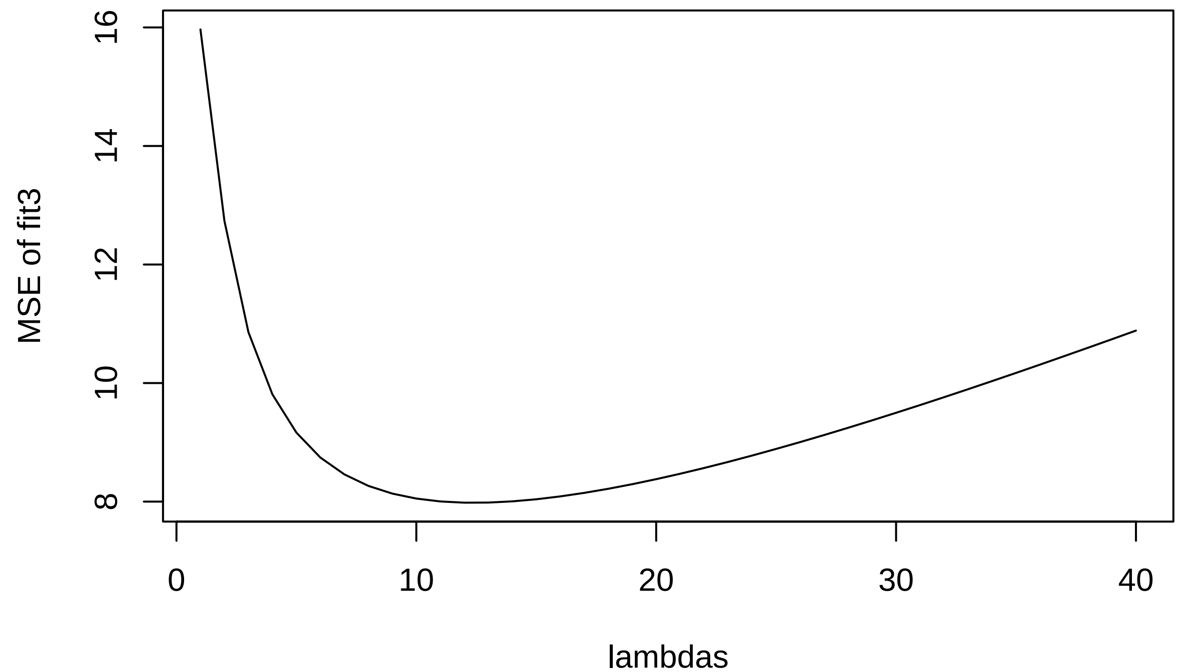
```
##                     cyl        disp          hp        drat          wt
## 16.80983460 -0.08921105  0.02537022 -0.03223653  0.13090241 -6.28117422
##        qsec          vs          am        gear        carb
##  0.90843660  0.35961234  0.60090547  1.08433927  0.45213358
```

```
y.pred3 <- matrix(NA, 6, 40)

# Because I cannot guarantee the order of predictors.
for (i in 1:40) {
  y.pred3[,i] <- test3_predictors[, c("const")] * coef(fit3)[i,1] +
    test3_predictors[, c("X3.disp")] * coef(fit3)[i,c("disp")] +
    test3_predictors[, c("X3.carb")] * coef(fit3)[i,c("carb")] +
    test3_predictors[, c("X3.gear")] * coef(fit3)[i,c("gear")] +
    test3_predictors[, c("X3.wt")] * coef(fit3)[i,c("wt")] +
```

```
      test3_predictors[, c("X3.drat")] * coef(fit3)[i,c("drat")] +
      test3_predictors[, c("X3.hp")] * coef(fit3)[i,c("hp")] +
      test3_predictors[, c("X3.qsec")] * coef(fit3)[i,c("qsec")] +
      test3_predictors[, c("X3.cyl")] * coef(fit3)[i,c("cyl")] +
      test3_predictors[, c("X3.vs")] * coef(fit3)[i,c("vs")] +
      test3_predictors[, c("X3.am")] * coef(fit3)[i,c("am")]
}

errors3 <- (y.pred3 - test3[,c("X3.mpg")])^2
colMeans(errors3) #MSE
```

```
##  [1] 15.966309 12.735619 10.859081  9.809523  9.164629  8.743417  8.459107
##  [8]  8.265285  8.134897  8.051065  8.002679  7.982094  7.983852  8.003939
## [15]  8.039319  8.087648  8.147074  8.216115  8.293560  8.378412  8.469833
## [22]  8.567118  8.669662  8.776945  8.888514  9.003973  9.122973  9.245202
## [29]  9.370385  9.498272  9.628639  9.761284  9.896022 10.032684 10.171115
## [36] 10.311176 10.452733 10.595667 10.739866 10.885226
```

```
plot(colMeans(errors3), type="l", xlab="lambdas",ylab="MSE of fit3")
```



Now I use the test data to predict values for y for model 4

```
test4_predictors <- subset(as.matrix(cbind(const=1, test4)), select = -X4.mpg)
coef(fit4)[1,]
```

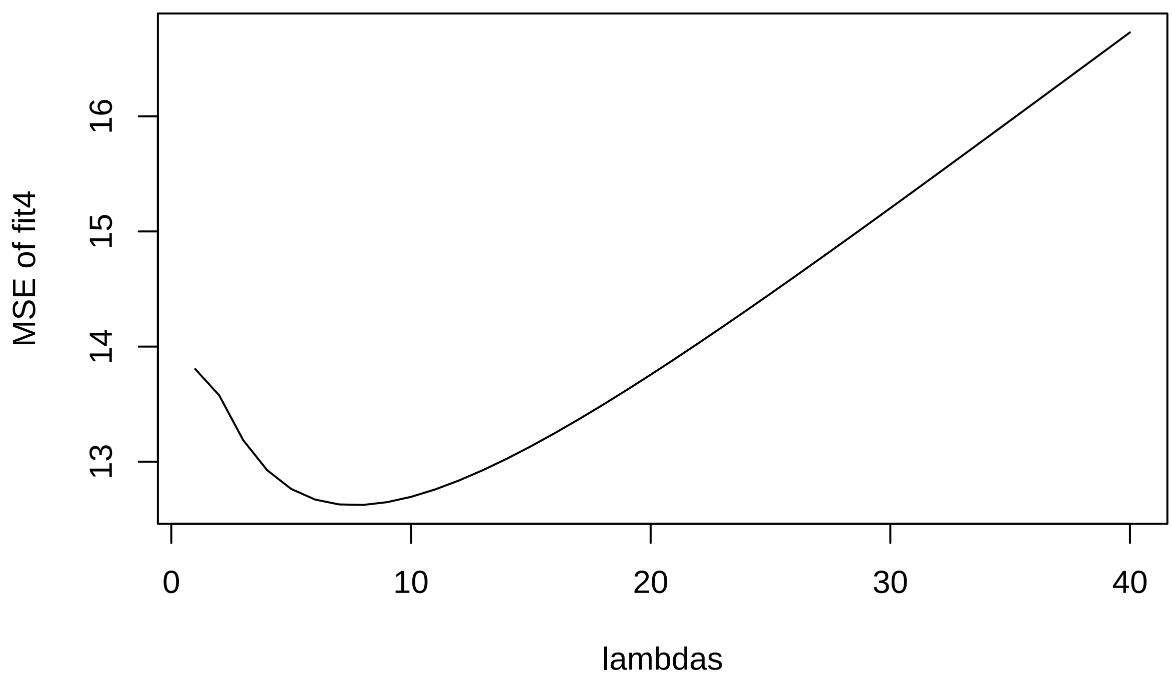
```
##                       cyl          disp            hp          drat            wt
## 35.690028389 -0.892149026  0.002766237 -0.011570003 -0.401256940 -2.911487404
##         qsec           vs            am          gear          carb
##  0.042087395 -0.575494216 -0.103730418  0.849509301 -0.625708646
```

```
y.pred4 <- matrix(NA, 6, 40)

# Because I cannot guarantee the order of predictors.
for (i in 1:40) {
```

```
  y.pred4[,i] <- test4_predictors[, c("const")] * coef(fit4)[i,1] +
    test4_predictors[, c("X4.disp")] * coef(fit4)[i,c("disp")] +
    test4_predictors[, c("X4.carb")] * coef(fit4)[i,c("carb")] +
    test4_predictors[, c("X4.gear")] * coef(fit4)[i,c("gear")] +
    test4_predictors[, c("X4.wt")] * coef(fit4)[i,c("wt")] +
    test4_predictors[, c("X4.drat")] * coef(fit4)[i,c("drat")] +
    test4_predictors[, c("X4.hp")] * coef(fit4)[i,c("hp")] +
    test4_predictors[, c("X4.qsec")] * coef(fit4)[i,c("qsec")] +
    test4_predictors[, c("X4.cyl")] * coef(fit4)[i,c("cyl")] +
    test4_predictors[, c("X4.vs")] * coef(fit4)[i,c("vs")] +
    test4_predictors[, c("X4.am")] * coef(fit4)[i,c("am")]
}

errors4 <- (y.pred4 - test4[,c("X4.mpg")])^2
colMeans(errors4) #MSE
```

```
##  [1] 13.80449 13.57582 13.18723 12.92606 12.76361 12.67108 12.62899 12.62435
##  [9] 12.64827 12.69446 12.75835 12.83653 12.92641 13.02595 13.13358 13.24801
## [17] 13.36820 13.49331 13.62262 13.75556 13.89161 14.03035 14.17143 14.31452
## [25] 14.45936 14.60570 14.75334 14.90209 15.05180 15.20232 15.35351 15.50527
## [33] 15.65749 15.81008 15.96295 16.11603 16.26924 16.42253 16.57584 16.72912
```

```
plot(colMeans(errors4), type="l", xlab="lambdas",ylab="MSE of fit4")
```



Now I use the test data to predict values for y for model 5

```
test5_predictors <- subset(as.matrix(cbind(const=1, test5)), select = -X5.mpg)
coef(fit5)[1,]
```

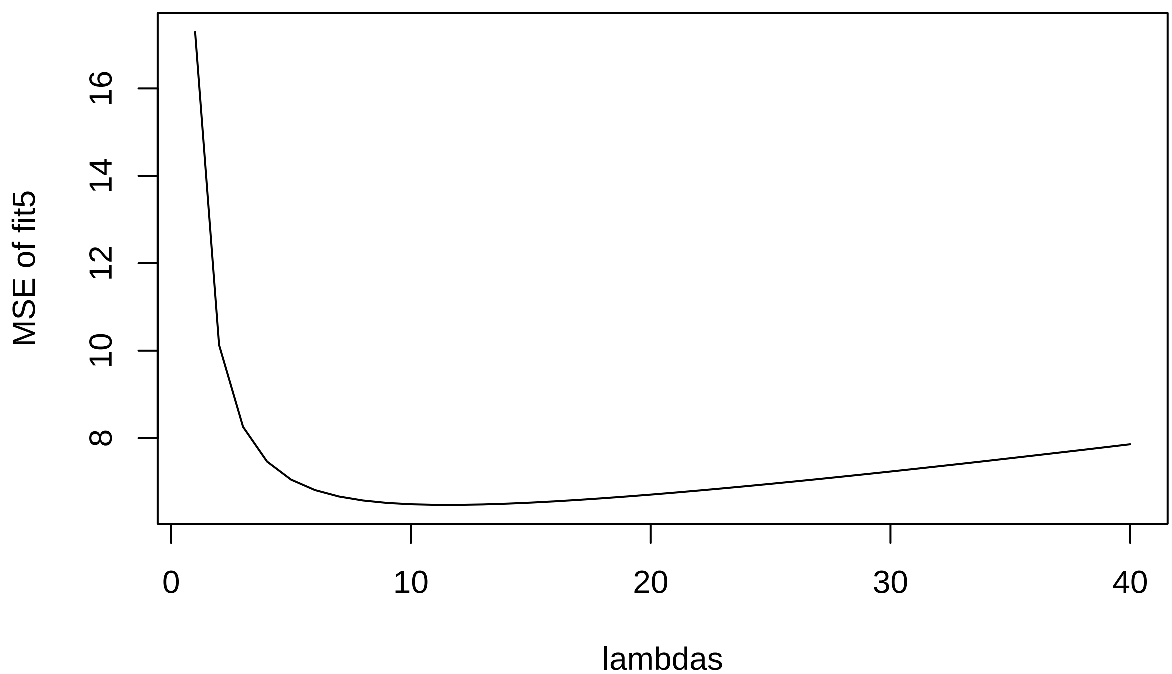
```
##                         cyl          disp            hp          drat 
## -19.965545027   1.233166567   0.005712027  -0.005020321   3.879366903 
##            wt          qsec            vs            am          gear 
##  -2.314574694   1.061551698   0.149021309   2.802755469   2.662077907 
##          carb
```

```
##  -1.585551631
```

```r
y.pred5 <- matrix(NA, 6, 40)

# Because I cannot guarantee the order of predictors.
for (i in 1:40) {
   y.pred5[,i] <- test5_predictors[, c("const")] * coef(fit5)[i,1] +
      test5_predictors[, c("X5.disp")] * coef(fit5)[i,c("disp")] +
      test5_predictors[, c("X5.carb")] * coef(fit5)[i,c("carb")] +
      test5_predictors[, c("X5.gear")] * coef(fit5)[i,c("gear")] +
      test5_predictors[, c("X5.wt")] * coef(fit5)[i,c("wt")] +
      test5_predictors[, c("X5.drat")] * coef(fit5)[i,c("drat")] +
      test5_predictors[, c("X5.hp")] * coef(fit5)[i,c("hp")] +
      test5_predictors[, c("X5.qsec")] * coef(fit5)[i,c("qsec")] +
      test5_predictors[, c("X5.cyl")] * coef(fit5)[i,c("cyl")] +
      test5_predictors[, c("X5.vs")] * coef(fit5)[i,c("vs")] +
      test5_predictors[, c("X5.am")] * coef(fit5)[i,c("am")]
}

errors5 <- (y.pred5 - test5[,c("X5.mpg")])^2
colMeans(errors5) #MSE
```

```
##  [1] 17.292255 10.129389  8.256298  7.463253  7.049912  6.810164  6.663462
##  [8]  6.572285  6.516912  6.486028  6.472687  6.472391  6.482103  6.499693
## [15]  6.523627  6.552767  6.586251  6.623413  6.663728  6.706779  6.752224
## [22]  6.799785  6.849230  6.900362  6.953014  7.007045  7.062331  7.118762
## [29]  7.176244  7.234692  7.294031  7.354192  7.415115  7.476743  7.539025
## [36]  7.601916  7.665371  7.729352  7.793821  7.858744
```

```r
plot(colMeans(errors5), type="l", xlab="lambdas",ylab="MSE of fit5")
```

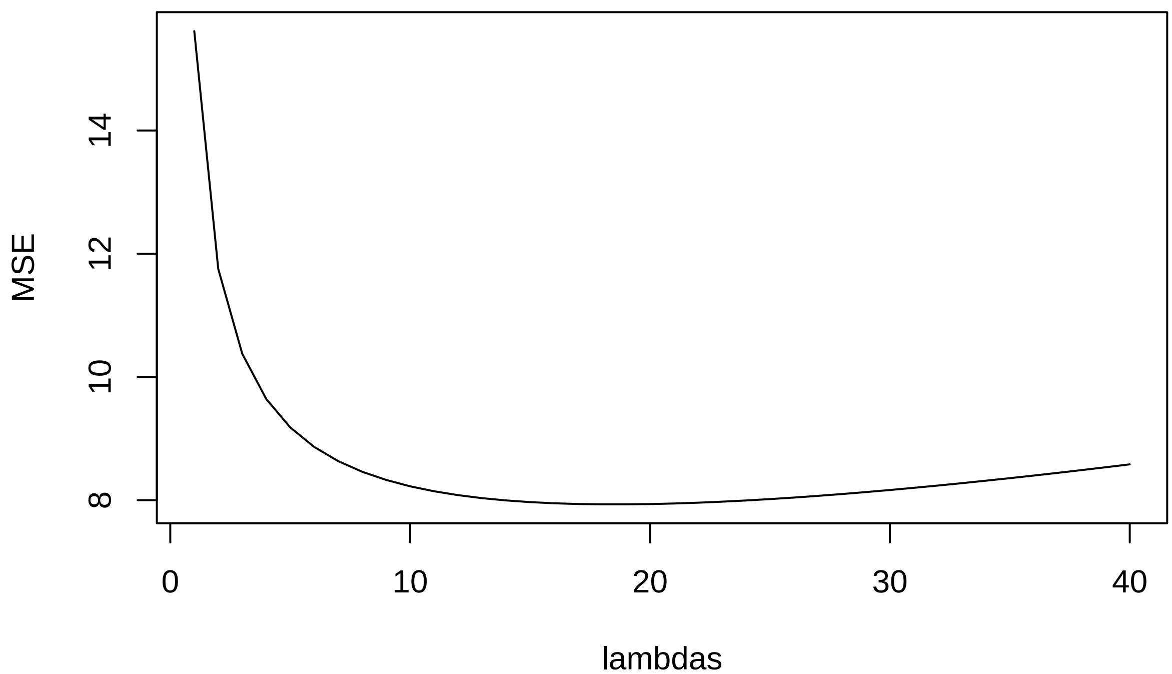


From these 5-fold CV models, I would choose lambda value around 19 because that is where the MSE is the lowest for all 5 models on average. I confirm the choice below in averaging the errors across models.

```r
mean_errors <- colMeans(rbind(errors1, errors2, errors3, errors4, errors5))
mean_errors
```

```
##  [1] 15.613416 11.753366 10.376873  9.642884  9.181586  8.864467  8.634300
##  [8]  8.461389  8.328631  8.225398  8.144697  8.081707  8.032983  7.995985
## [15]  7.968793  7.949921  7.938199  7.932689  7.932627  7.937386  7.946439
## [22]  7.959344  7.975725  7.995259  8.017663  8.042695  8.070138  8.099802
## [29]  8.131519  8.165136  8.200517  8.237539  8.276091  8.316071  8.357386
## [36]  8.399949  8.443684  8.488516  8.534379  8.581210
```

```r
plot(mean_errors, type="l", xlab="lambdas",ylab="MSE")
```




```r
which.min(mean_errors)
```

```
## [1] 19
```

The GCV result chooses 15 as the penalty level (according to the lecture note). This is not too far off from what we choose from our 5-fold CV method.

- Use the `cv.glmnet()` function from the `glmnet` package to perform a 5-fold cross-validation using their built-in feature. Produce the cross-validation error plot against λ values. Report the `lambda.min` and `lambda.1se` selected λ value.

```r
install.packages("glmnet",repos = "http://cran.us.r-project.org")
```

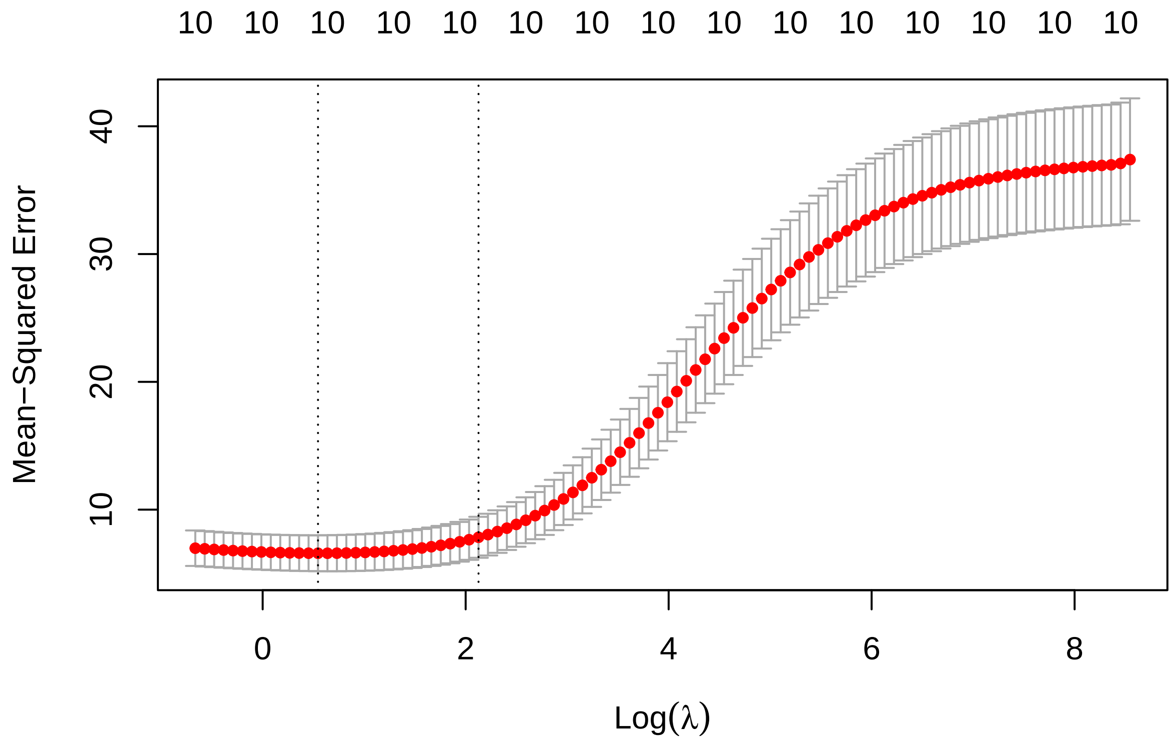
```
##
## The downloaded binary packages are in
##  /var/folders/9c/3_mgdyf12z7dvb8rt4d60nt80000gn/T//RtmpKcMp4p/downloaded_packages
```

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

```r
fit_glmnet = cv.glmnet(x = data.matrix(mtcars[, -1]), y = mtcars$mpg, nfolds = 5, alpha = 0)
plot(fit_glmnet)
```



According to the plot, MSE is the lowest when Log(lambda) is 0 to 1 or lamba is 1 to e. Below I extracted lambda.min and lambda.1se as well as the coefficients of the model using these penalty values.

```
fit_glmnet$lambda.min
```

```
## [1] 1.725064
```

```
fit_glmnet$lambda.1se
```

```
## [1] 8.388297
```

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

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept) 21.23747093
## cyl         -0.35038148
## disp        -0.00476129
## hp          -0.01204417
## drat         1.04051063
## wt          -1.38907664
## qsec         0.18081288
## vs           0.68926400
## am           1.77836781
## gear         0.55941083
## carb        -0.60479256
```

```
coef(fit_glmnet, s = "lambda.1se")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) 19.958741467
## cyl         -0.366245210
```

```
## disp        -0.005307137
## hp          -0.009826554
## drat         0.996971267
## wt          -0.896377898
## qsec         0.152126013
## vs           0.873268719
## am           1.214907797
## gear         0.501485399
## carb        -0.383276276
```