RidgeRegression

Group 2-07

Setting up

packages needed: "leaps", "ISLR"(for data), "glmnet" (for regression models)

```
options(warn=-1)   # Supress warning messages
installIfAbsentAndLoad <- function(neededVector) {
  for(thispackage in neededVector) {
    if( ! require(thispackage, character.only = T) )
    { install.packages(thispackage)}
    require(thispackage, character.only = T)
  }
}

# class contains the knn() function, KNN contains the
# knn.reg() function, combinat contains the combn() function
needed <- c("leaps", "ISLR", "glmnet")
installIfAbsentAndLoad(needed)
```

```
## Loading required package: leaps

## Loading required package: ISLR

## Loading required package: glmnet

## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-13
```

We will use the **glmnet** package in order to perform ridge regression and the lasso.
The main function in this package is *glmnet()*, which can be used to fit ridge regression models, lasso models, and more.
This function has slightly different syntax from other model-fitting functions that we have encountered thus far in this book. In particular, we must pass in an x matrix as well as a y vector, and we do not use the $y \sim x$ syntax. We will now perform ridge regression and the lasso in order to predict Salary on the Hitters data.

The *model.matrix()* function is particularly useful for creating x; not only does it produce a matrix corresponding to the 19 predictors but it also automatically transforms any qualitative variables into dummy variables.
The latter property is important because *glmnet()* can only take numerical, quantitative inputs.

```
Hitters <- na.omit(Hitters)
str(Hitters)
```

```
## 'data.frame':    263 obs. of  20 variables:
##  $ AtBat    : int  315 479 496 321 594 185 298 323 401 574 ...
##  $ Hits     : int  81 130 141 87 169 37 73 81 92 159 ...
##  $ HmRun    : int  7 18 20 10 4 1 0 6 17 21 ...
##  $ Runs     : int  24 66 65 39 74 23 24 26 49 107 ...
##  $ RBI      : int  38 72 78 42 51 8 24 32 66 75 ...
##  $ Walks    : int  39 76 37 30 35 21 7 8 65 59 ...
```

```
##  $ Years    : int  14 3 11 2 11 2 3 2 13 10 ...
##  $ CAtBat   : int  3449 1624 5628 396 4408 214 509 341 5206 4631 ...
##  $ CHits    : int  835 457 1575 101 1133 42 108 86 1332 1300 ...
##  $ CHmRun   : int  69 63 225 12 19 1 0 6 253 90 ...
##  $ CRuns    : int  321 224 828 48 501 30 41 32 784 702 ...
##  $ CRBI     : int  414 266 838 46 336 9 37 34 890 504 ...
##  $ CWalks   : int  375 263 354 33 194 24 12 8 866 488 ...
##  $ League   : Factor w/ 2 levels "A","N": 2 1 2 2 1 2 1 2 1 1 ...
##  $ Division : Factor w/ 2 levels "E","W": 2 2 1 1 2 1 2 2 1 1 ...
##  $ PutOuts  : int  632 880 200 805 282 76 121 143 0 238 ...
##  $ Assists  : int  43 82 11 40 421 127 283 290 0 445 ...
##  $ Errors   : int  10 14 3 4 25 7 9 19 0 22 ...
##  $ Salary   : num  475 480 500 91.5 750 ...
##  $ NewLeague: Factor w/ 2 levels "A","N": 2 1 2 2 1 1 1 2 1 1 ...
##  - attr(*, "na.action")=Class 'omit'  Named int [1:59] 1 16 19 23 31 33 37 39 40 42 ...
##   .. ..- attr(*, "names")= chr [1:59] "-Andy Allanson" "-Billy Beane" "-Bruce Bochte" "-Bob Boone" .
```

```
x <- model.matrix(Salary ~ ., Hitters)[, -1]     #We omit the intercept
y <- Hitters$Salary
#install.packages("glmnet")
```

Ridge Regression

- The glmnet() function
  The **glmnet()** function has an α argument that determines what type of model is fit. If $\alpha = 0$ then a ridge regression model is fit, and if $\alpha = 1$ (the default) then a lasso model is fit. In this lab we want to fit a ridge regression model so we need to set it to 0.
  By default the *glmnet()* function performs ridge regression for an automatically selected range of lamda values. However, here we have chosen to implement the function over a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit. As we will see, we can also compute model fits for a particular value of λ that is not one of the original grid values.

Note that by default, the *glmnet()* function standardizes the variables so that they are on the same scale. To turn off this default setting, use the argument **standardize=FALSE**.
Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by *coef()*. In this case, it is a 20×100 matrix, with 20 rows (19 predictors plus an intercept) and 100 columns (one for each value of λ).

Divide the interval from 10 to -2 into 100 partitions - make these the exponents of 10, so the *grid* vector goes from 10^{10} to 0.01 in 100 (diminishing) steps.

```
grid <- 10 ^ seq(10, -2, length=100)
ridge.mod <- glmnet(x,y,alpha=0,lambda=grid)
str(ridge.mod$beta)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:1900] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:101] 0 19 38 57 76 95 114 133 152 171 ...
##   ..@ Dim     : int [1:2] 19 100
##   ..@ Dimnames:List of 2
##   .. ..$ : chr [1:19] "AtBat" "Hits" "HmRun" "Runs" ...
##   .. ..$ : chr [1:100] "s0" "s1" "s2" "s3" ...
##   ..@ x       : num [1:1900] 5.44e-08 1.97e-07 7.96e-07 3.34e-07 3.53e-07 ...
```

```
##   ..@ factors : list()
```

```r
dim(coef(ridge.mod))
```

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

We expect the coefficient estimates to be much smaller, in terms of l2 norm, when a large value of λ is used, as compared to a small one.

```r
# These are the coefficients when lamda = 10,000,000,000, along with their l2 norm:
ridge.mod$lambda[1]
```

```
## [1] 1e+10
```

```r
coef(ridge.mod)[,1]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  5.359257e+02  5.443467e-08  1.974589e-07  7.956523e-07  3.339178e-07
##           RBI         Walks         Years        CAtBat         CHits
##  3.527222e-07  4.151323e-07  1.697711e-06  4.673743e-09  1.720071e-08
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  1.297171e-07  3.450846e-08  3.561348e-08  3.767877e-08 -5.800263e-07
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -7.807263e-06  2.180288e-08  3.561198e-09 -1.660460e-08 -1.152288e-07
```

```r
sqrt(sum(coef(ridge.mod)[-1,1]^2))
```

```
## [1] 8.080244e-06
```

```r
# These are the coefficients when lamda = 11,498, along with their l2 norm:
ridge.mod$lambda[50]
```

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

```r
coef(ridge.mod)[,50]
```

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

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

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

Note the much larger l2 norm of the coefficients associated with the smaller value of λ.
We can use the *predict()* function for a number of purposes. For instance, we can obtain the ridge regression coefficients for a new value of λ, say 50:

```r
predict(ridge.mod,s=50,type="coefficients")[1:20,]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00
##           RBI         Walks         Years        CAtBat         CHits
##  8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01
```

```
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

We now split the samples into a training set and a test set in order to estimate the test error of ridge regression and the lasso.

```
set.seed(1)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test <- y[test]
```

Next we fit a ridge regression model on the training set, and evaluate its MSE on the test set, using lamda = 4. Note the use of the predict() function again. This time we get predictions for a test set, by replacing type="coefficients" with the newx argument.

```
ridge.mod <- glmnet(x[train,],y[train],alpha=0,lambda=grid, thresh=1e-12)
ridge.pred <- predict(ridge.mod,s=4,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 101036.8
```

The test MSE is 101037. Note that if we had instead simply fit a model with just an intercept, we would have predicted each test observation using the mean of the training observations. In that case, we could compute the test set MSE like this:

```
mean((mean(y[train])-y.test)^2)
```

```
## [1] 193253.1
```

```
# We could also get the same result by fitting a ridge regression model with
# a very large value of lamda. Note that 1e10 means 10^10.
ridge.pred <- predict(ridge.mod,s=1e10,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 193253.1
```

So fitting a ridge regression model with $\lambda = 4$ leads to a much lower test MSE than fitting a model with just an intercept. We now check whether there is any benefit to performing ridge regression with $\lambda = 4$ instead of just performing least squares regression. Recall that least squares is simply ridge regression with $\lambda = 0$
Note: In order for *glmnet()* to yield the exact least squares coefficients when $\lambda = 0$, we use the argument *exact=TRUE* when calling the predict() function. Otherwise, the predict() function will interpolate over the grid of lamda values used in fitting the glmnet() model, yielding approximate results. When we use exact=T, there remains a slight discrepancy in the third decimal place between the output of glmnet() when $\lambda = 0$ and the output of lm(); this is due to numerical approximation on the part of glmnet().

```
ridge.pred <- predict(ridge.mod,s=0,newx=x[test,],exact=T,x=x[train,],y=y[train])
mean((ridge.pred-y.test)^2)
```

```
## [1] 114783.1
```

```
lm(y~x, subset=train)
```

```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)       xAtBat        xHits       xHmRun        xRuns
##   299.42849     -2.54027      8.36682     11.64512     -9.09923
```

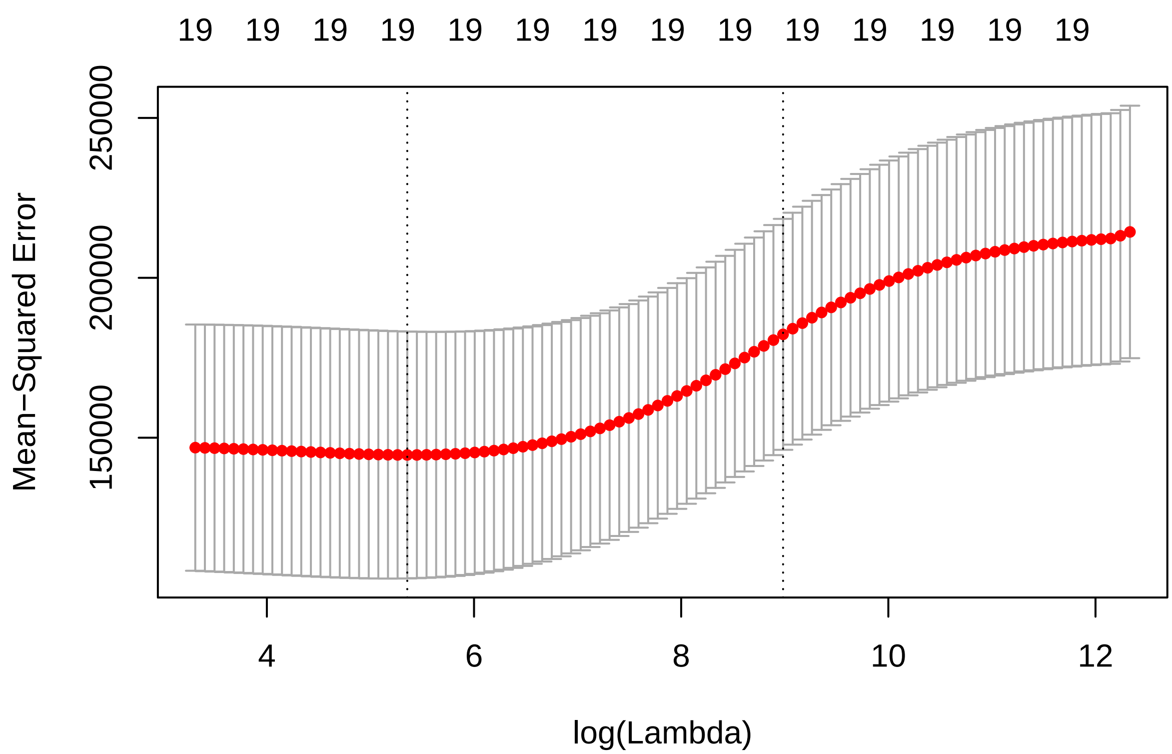
```
##          xRBI        xWalks        xYears       xCAtBat        xCHits
##       2.44105       9.23440     -22.93673      -0.18154      -0.11598
##       xCHmRun        xCRuns         xCRBI       xCWalks      xLeagueN
##      -1.33888       3.32838       0.07536      -1.07841      59.76065
##    xDivisionW      xPutOuts      xAssists       xErrors   xNewLeagueN
##     -98.86233       0.34087       0.34165      -0.64207      -0.67442
```

```r
predict(ridge.mod,s=0,exact=T,type="coefficients",x=x[train,],y=y[train])[1:20,]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## 299.42883596   -2.54014665    8.36611719   11.64400720   -9.09877719
##           RBI         Walks         Years        CAtBat         CHits
##    2.44152119    9.23403909  -22.93584442   -0.18160843   -0.11561496
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##   -1.33836534    3.32817777    0.07511771   -1.07828647   59.76529059
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -98.85996590    0.34086400    0.34165605   -0.64205839   -0.67606314
```

In general, if we want to fit a (unpenalized) least squares model, then we should use the lm() function, since that function provides more useful outputs, such as standard errors and p-values for the coefficients.
Also, instead of arbitrarily choosing lamda = 4, it would be better to use cross-validation to choose the tuning parameter lamda. We can do this using the built-in cross-validation function, cv.glmnet(). By default, the function performs ten-fold cross-validation, though this can be changed using the argument nfolds. Note that we set a random seed first.

```r
set.seed(1)
cv.out <- cv.glmnet(x[train,],y[train],alpha=0)
plot(cv.out)
```



```r
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 211.7416
```

Therefore, we see that the value of λ that results in the smallest cross validation error is 212. What is the test MSE associated with this value of λ?

```r
ridge.pred <- predict(ridge.mod,s=bestlam,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 96015.51
```

This represents a further improvement over the test MSE that we got using $\lambda = 4$. Finally, we refit our ridge regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

`r out <- glmnet(x,y,alpha=0) predict(out,type="coefficients",s=bestlam)[1:20,]`

`## (Intercept) AtBat Hits HmRun Runs ## 9.88487157 0.03143991 1.00882875 0.13927624 1.11320781 ## RBI Walks Years CAtBat CHits ## 0.87318990 1.80410229 0.13074381 0.01113978 0.06489843 ## CHmRun CRuns CRBI CWalks LeagueN ## 0.45158546 0.12900049 0.13737712 0.02908572 27.18227535 ## DivisionW PutOuts Assists Errors NewLeagueN ## -91.63411299 0.19149252 0.04254536 -1.81244470 7.21208390`
As expected, none of the coefficients are zero.
Ridge regression does **not** perform variable selection!