

6.6 Lab: Ridge Regression and the Lasso

Zongyi Liu

2023-05-31

6.6 Ridge Regression and the Lasso

Here we will use `glmnet()` package

`model.matrix()` is helpful, because it can - produce a matrix corresponding to predictors - utomatically transforms any qualitative variables into dummy variables

```
library(ISLR)
Hitters=na.omit(Hitters)
x=model.matrix(Salary~.,Hitters)[,-1]
y=Hitters$Salary
```

6.1.1 Ridge Regression

In the `glmnet()` function, there is an `alpha` argument, if `alpha=0` then a ridge regression model is fit, and if `alpha=1` then a lasso model is fit.

```
library(glmnet)
```

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

```
## Loaded glmnet 4.1-7
```

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

By default the `glmnet()` function performs ridge regression for an automatically selected range of λ values

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

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

We expect the coefficient estimates to be much smaller, in terms of l_2 norm, when a large value of λ is used, as compared to when a small value of λ is used.

```
ridge.mod$lambda[50]
```

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

```
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
```

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

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

We can use the `predict()` function for a number of purposes, for example, we can obtain the ridge regression coefficients for a new value of λ , say 50.

```
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 can then split the samples into a training and a test set to test the errors of ridge regression and lasso.

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

Then we fit a ridge regression model on the training set, and evaluate the MSE on it

```
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] 142199.2
```

If we have simply fit a model with just an intercept, we would have predicted each test observation using the mean of the training observations.

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

```
## [1] 224669.9
```

We could also get the same result by fitting a ridge regression model with a very large value of λ :

```
ridge.pred=predict(ridge.mod,s=1e10,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 224669.8
```

So we can say that fitting a ridge regression model with $\lambda = 4$ leads to a much lower test MSE than fitting a model with just an intercept.

```
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] 168588.6
```

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

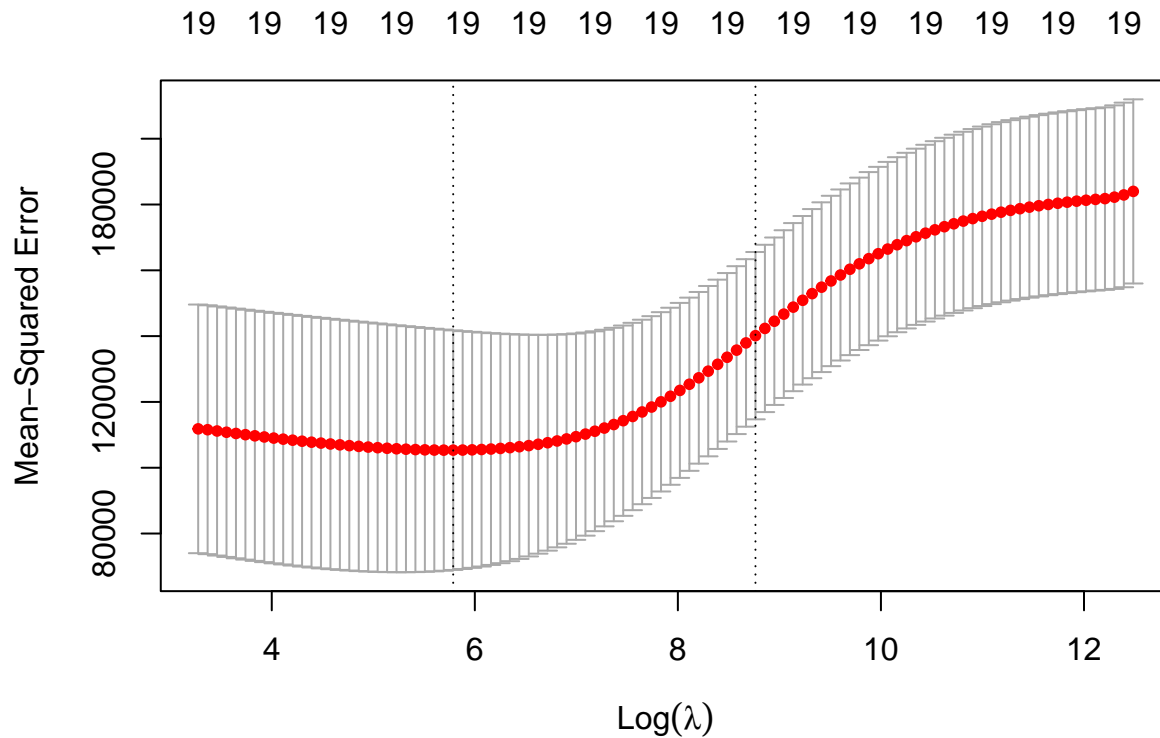
```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)      xAtBat      xHits      xHmRun      xRuns      xRBI
##    274.0145    -0.3521    -1.6377     5.8145     1.5424     1.1243
##      xWalks      xYears      xCAtBat      xCHits      xCHmRun      xCRuns
##     3.7287    -16.3773    -0.6412     3.1632     3.4008    -0.9739
##      xCRBI      xCWalks      xLeagueN      xDivisionW      xPutOuts      xAssists
##    -0.6005     0.3379    119.1486   -144.0831     0.1976     0.6804
##      xErrors      xNewLeagueN
##    -4.7128    -71.0951
```

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

```
## (Intercept)      AtBat      Hits      HmRun      Runs      RBI
## 274.0200994  -0.3521900  -1.6371383  5.8146692  1.5423361  1.1241837
##      Walks      Years      CAtBat      CHits      CHmRun      CRuns
## 3.7288406  -16.3795195  -0.6411235  3.1629444  3.4005281  -0.9739405
##      CRBI      CWalks      LeagueN      DivisionW      PutOuts      Assists
## -0.6003976  0.3378422  119.1434637 -144.0853061  0.1976300  0.6804200
##      Errors      NewLeagueN
## -4.7127879  -71.0898914
```

Also, instead of arbitrarily choosing $\lambda = 4$, it would be better to use cross-validation to choose the tuning parameter λ . We can do this using the built-in cross-validation function, `cv.glmnet()`.

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



```
bestlam=cv.out$lambda.min
bestlam
```

```
## [1] 326.0828
```

Here we can see that the value of λ associated with the smallest cv-error is 212, then we will try to find the test MSE.

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

```
## [1] 139856.6
```

This shows the improvement of our model, and we will choose it as our final model:

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

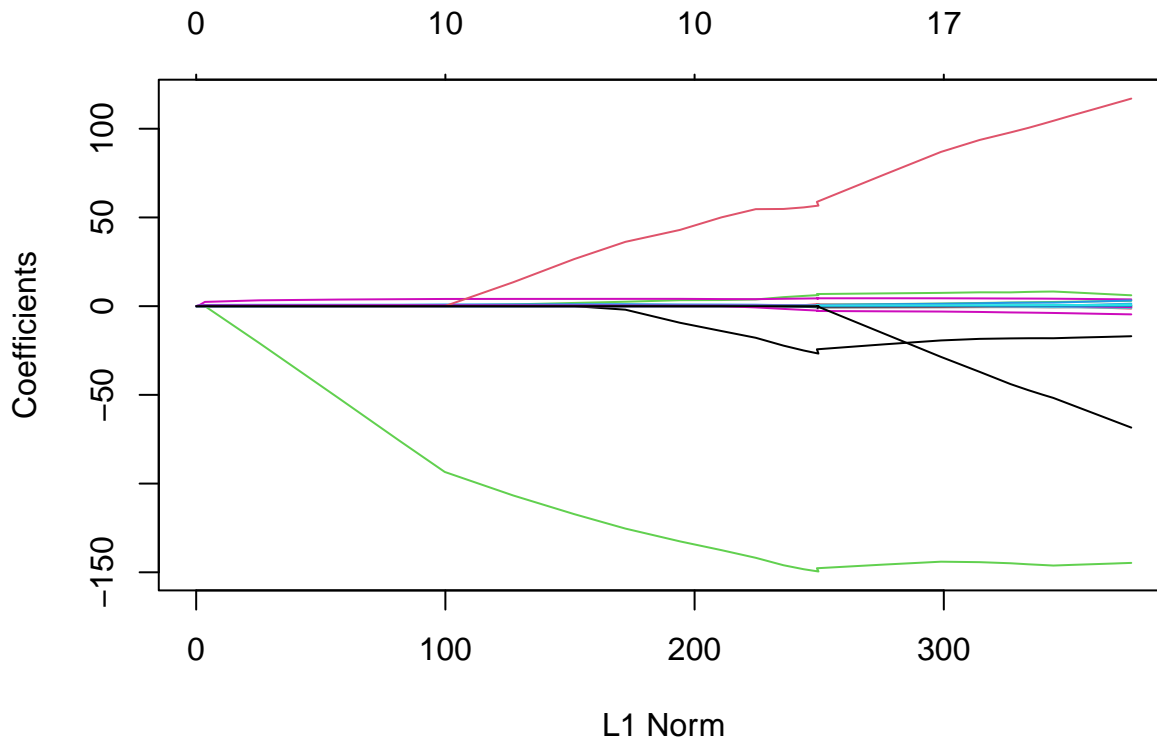
```
## (Intercept)      AtBat      Hits      HmRun      Runs      RBI
## 15.44383120  0.07715547  0.85911582  0.60103106  1.06369007  0.87936105
##      Walks      Years      CAtBat      CHits      CHmRun      CRuns
##  1.62444617  1.35254778  0.01134999  0.05746654  0.40680157  0.11456224
##      CRBI      CWalks      LeagueN      DivisionW      PutOuts      Assists
##  0.12116504  0.05299202  22.09143197 -79.04032656  0.16619903  0.02941950
##      Errors      NewLeagueN
## -1.36092945  9.12487765
```

6.6.2 Lasso Regression

We will use the same function to perform lasso regression

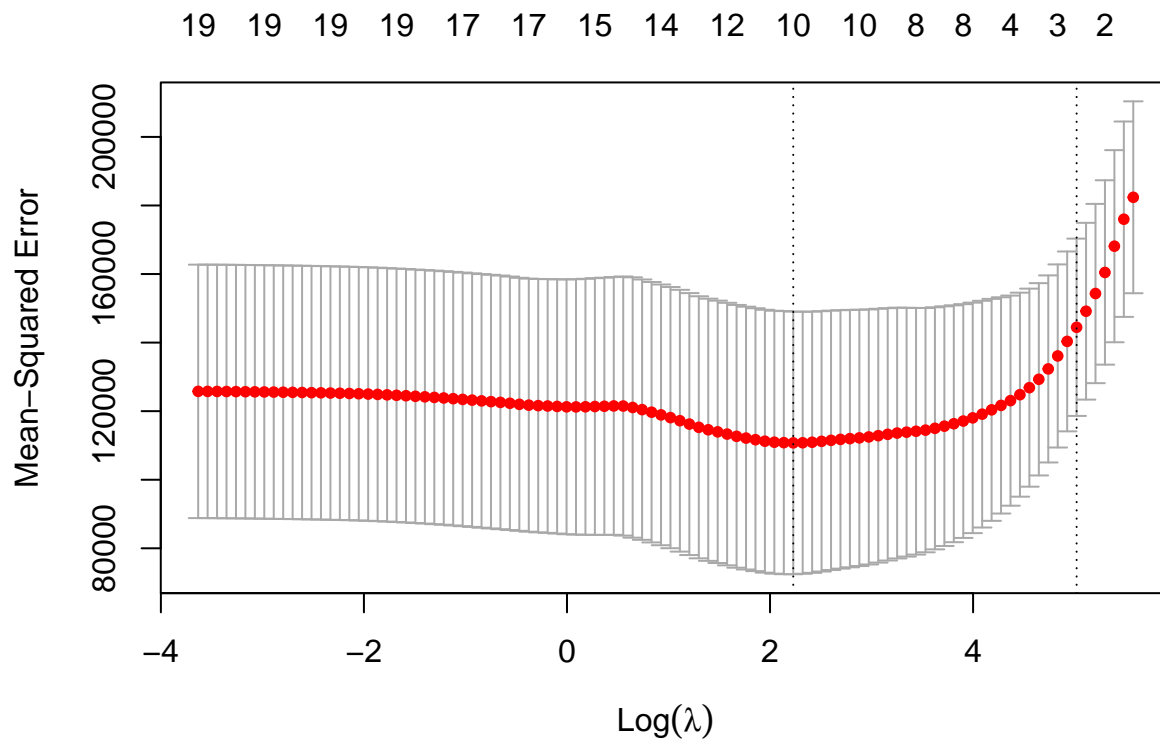
```
lasso.mod=glmnet(x[train,],y[train],alpha=1,lambda=grid)
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



Then we have

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



```
bestlam=cv.out$lambda.min
lasso.pred=predict(lasso.mod,s=bestlam,newx=x[test,])
mean((lasso.pred-y.test)^2)
```

```
## [1] 143673.6
```

This is substantially lower than the test set MSE of the null model and of least squares, and very similar to the test MSE of ridge regression with λ chosen by cross-validation.