NBA 4920/6921 Lecture 14

Lasso Regression Application

Murat Unal

10/19/2021

```r
rm(list=ls())
options(digits = 3, scipen = 999)
library(ggplot2)
library(tidyverse)
library(ISLR)
library(jtools)
library(caret)
library(leaps)
library(glmnet)
Hitters <- ISLR::Hitters
Hitters <- na.omit(Hitters)
set.seed(2)
```

Ridge Regression

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

- 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.

- The **glmnet()** engine has an `alpha` argument that determines what type of model is fit.
- If `alpha=0` then a ridge regression model is fit, and if `alpha=1` then a lasso model is fit. We first fit a ridge regression model.
- By default, the **glmnet()** engine standardizes the variables so that they are on the same scale.

- **glmnet()** will fit ridge models across a wide range of λ values by default, 100 λ values that are data derived
- We could also run the engine for a grid of 100 values ranging from high to low λ

```r
grid=10^seq(10,-2,length=100)
grid[1]
```

```
[1] 10000000000
```

```r
grid[100]
```

```
[1] 0.01
```

- For regression problems set `familty="gaussian"`, and `familty="binomial"` for classification

```
ridge.mod=glmnet(x,y,alpha=0,lambda=grid,family="gaussian")
names(ridge.mod)
```

```
 [1] "a0"        "beta"      "df"        "dim"       "lamb
 [7] "nulldev"   "npasses"   "jerr"      "offset"    "call
```

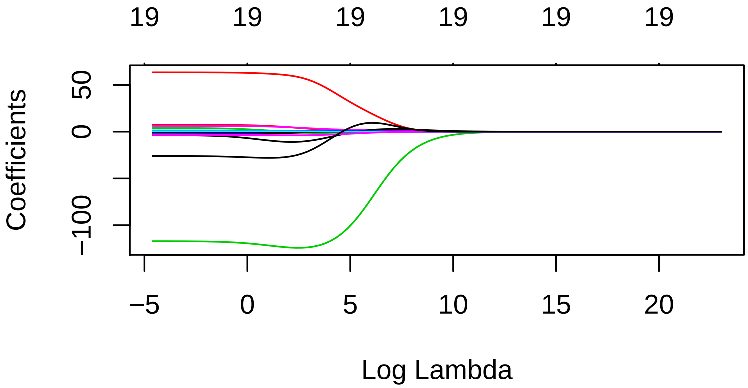
- Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()`.
- In this case, it is a $\mathbf{20 \times 100}$ matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ).

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

```
[1]  20 100
```

- As λgrows larger, our ridge regression coefficient magnitudes are more constrained.

```
plot(ridge.mod, xvar = "lambda")
```



```r
#Display 1st lambda value
ridge.mod$lambda[1]
```

```
[1] 10000000000
```

```r
# Display coefficients associated with
# 1st lambda value
coef(ridge.mod)[,1]
```

```
   (Intercept)          AtBat           Hits            H
535.92569352090  0.00000005443  0.00000019746  0.000000
           RBI          Walks          Years           CA
  0.00000035272  0.00000041513  0.00000169771  0.000000
        CHmRun          CRuns           CRBI           CW
  0.00000012972  0.00000003451  0.00000003561  0.000000
     DivisionW        PutOuts        Assists           Er
 -0.00000780726  0.00000002180  0.00000000356 -0.000000
```

- 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:
- If the desired λ value is not included in the initial fit, then `glmnet()` performs linear interpolation to make predictions for the desired λ value.
- Linear interpolation usually works fine, but we could change this by calling `exact=TRUE`. This way predictions are to be made at values of λ not included in the original fit and the model is refit before predictions are made.

- Let's check if $\lambda = 50$ is used in the original fit

```
any(ridge.mod$lambda==50)
```

```
[1] FALSE
```

- Let's see if there's any difference

```
coef.approax <- predict(ridge.mod, s = 50, type = "coeffic
                        exact = FALSE)
coef.exact <- predict(ridge.mod, s = 50, type = "coefficie
                        exact = TRUE, x=x,y=y)

cbind(coef.approax, coef.exact)
```

```
20 x 2 sparse Matrix of class "dgCMatrix"
                    1         1
(Intercept)  48.76610  48.272
AtBat        -0.35810  -0.353
Hits          1.96936   1.951
HmRun        -1.27825  -1.290
Runs          1.14589   1.156
```

- Let's now split the samples into a training set and a test set in order to estimate the test error of ridge regression

```
train=sample(1:nrow(x), 0.7*nrow(x))
```

- Next we fit a ridge regression model on the training set, and evaluate its RMSE on the test set, using $\lambda = 4$.
- 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)
ridge.pred.lambda4=predict(ridge.mod,s=4,newx=x[-train,])
rmse.ridge.lambda4 = sqrt(mean((ridge.pred.lambda4-
                                    y[-train])^2))
rmse.ridge.lambda4
```

```
[1] 296
```

- 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:

```
sqrt(mean((mean(y[train])-y[-train])^2))
```

```
[1] 405
```

- We could also get the same result by fitting a ridge regression model with a very large value of λ. Note that 1e10 means 10^{10}

```
ridge.pred.lambdabig=predict(ridge.mod,s=1e10,
                             newx=x[-train,])

rmse.ridge.lambdabig <- sqrt(mean((ridge.pred.lambdabig-
                                    y[-train])^2))

rmse.ridge.lambdabig
```

```
[1] 405
```

- So fitting a ridge regression model with $\lambda = 4$ leads to a much lower -train RMSE than fitting a model with just an intercept.
- Let's 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 $\lambda = 0$

```
ridge.pred.lambda0=predict(ridge.mod,s=0,exact = TRUE,
          x=x[train,],y=y[train],newx=x[-train,])

rmse.ridge.lambda0 <- sqrt(mean((ridge.pred.lambda0-
                                   y[-train])^2))

rmse.ridge.lambda0
```

```
[1] 300
```

- It looks like we are indeed improving over regular least-squares!

Cross-validation

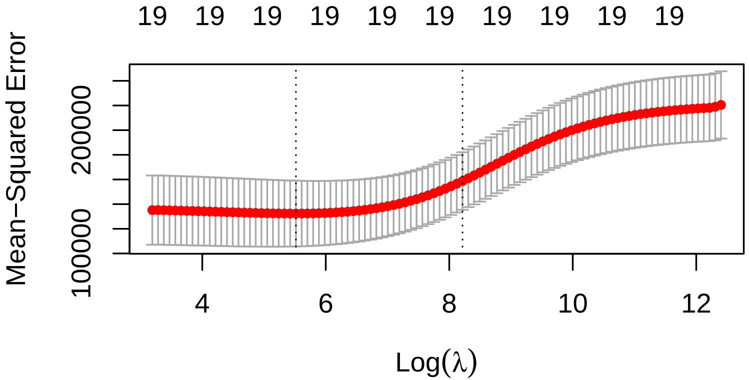
- Instead of arbitrarily choosing $\lambda = 4$, it would be better to use cross-validation to choose the tuning parameter λ.
- We can do this by writing our own function as we did before or we could just use the built-in cross-validation function, `cv.glmnet()`.
- By default, the function performs `10-fold` cross-validation, though this can be changed using the argument `folds`.
- We can define the loss used for cross-validation using `type.measure`

```
cv.out=cv.glmnet(x[train,],y[train],alpha=0,nfold=10,
                 type.measure="mse")
bestlam=cv.out$lambda.min
bestlam
```

```
[1] 249
```

- The first vertical dashed line gives the value of λ that gives minimum mean cross-validated error. The second is the λ that gives an MSE within one standard error of the smallest.

```
plot(cv.out)
```



- Finally, we obtain the coefficients from our fitted model using the value of λ chosen by cross-validation.
- As expected, none of the coefficients are zero—ridge regression does not perform variable selection!

```
predict(ridge.mod,type ="coefficients",s=bestlam,
        exact=TRUE, x=x[train,], y=y[train])
```

```
20 x 1 sparse Matrix of class "dgCMatrix"
                        1
(Intercept)      6.57462
AtBat           -0.00753
Hits             0.94106
HmRun            1.42067
Runs             1.21523
RBI              1.18983
Walks            2.16999
Years            0.43523
CAtBat           0.00963
CHits            0.05862
```

```r
final.pred = predict(ridge.mod, newx=x[-train,],s=bestlam,
                    exact=TRUE,x=x[train,],y=y[train])

rmse.ridge.lambdabest <- sqrt(mean((y[-train]-
                                     final.pred)^2))

rmse.ridge.lambdabest
```

```
[1] 292
```

- Let's compare all test RMSEs:

```
RMSE <- matrix(NA,ncol = 1, nrow = 8)
rownames(RMSE) <- c("rmse.ridge.lambdabig",
"rmse.ridge.lambda4","rmse.ridge.lambda0",
"rmse.ridge.lambdabest","rmse.lasso.lambda1se",
"rmse.lasso.lambdabest", "rmse.elnet.lambda1se",
"rmse.elnet.lambdabest")
RMSE[1:4,1] <- c(rmse.ridge.lambdabig,
rmse.ridge.lambda4,rmse.ridge.lambda0,
rmse.ridge.lambdabest)
RMSE
```

```
                      [,1]
rmse.ridge.lambdabig   405
rmse.ridge.lambda4     296
rmse.ridge.lambda0     300
rmse.ridge.lambdabest  292
rmse.lasso.lambda1se    NA
rmse.lasso.lambdabest   NA
rmse.elnet.lambda1se    NA
```

- Rdige regression for median λ value

```
median(grid)
```

```
[1] 10098
```

```
# Set s=median(grid) and make predictions
ridge.pred.lambdamedian=predict(ridge.mod,s=10098,
                                   newx=x[-train,])

rmse.ridge.lambdamedian<-sqrt(mean((ridge.pred.lambdamedian
                                          y[-train])^2))
rmse.ridge.lambdamedian
```

```
[1] 358
```

Lasso

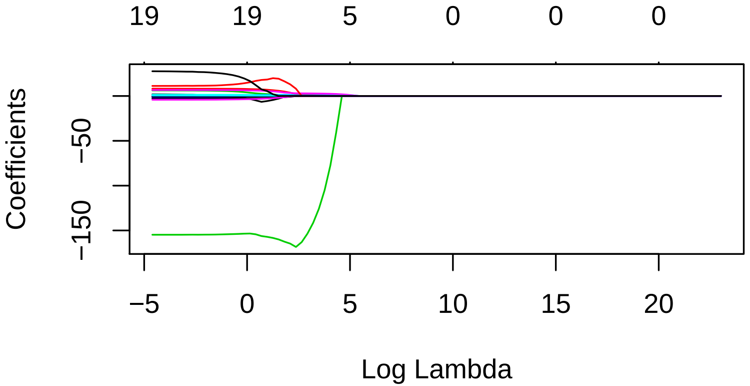
- We saw that ridge regression with a choice of λ can outperform least squares as well as the null model on the Hitters data set.
- We now ask whether the lasso can yield either a more accurate or a more interpretable model than ridge regression.
- We once again use the **glmnet()**; however, this time we use the argument $\alpha = 1$.
- Other than that change, we proceed just as we did in fitting a ridge model.

- Let's run a lasso regression using the previous grid values for λ and call it **lasso.mod**

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

- Let's plot the coefficients against the lambdas

```
plot(lasso.mod, xvar = "lambda")
```



- We can see from the coefficient plot that depending on the choice of tuning parameter, some of the coefficients will be exactly equal to zero.

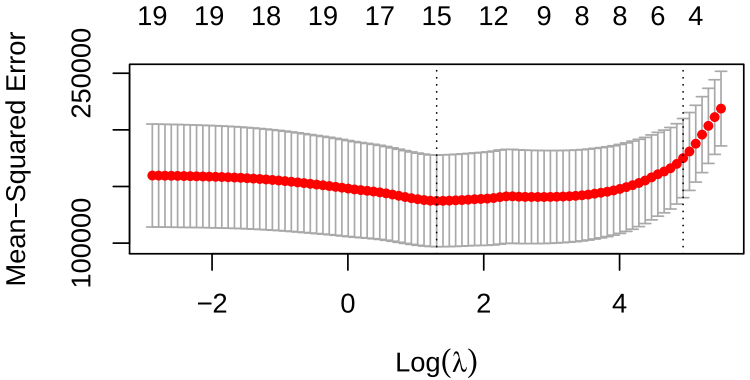
Cross-validation

- Let's do 10-fold cross-validation and save the output to **lasso.cv**.

```
lasso.cv=cv.glmnet(x[train,],y[train],alpha=1,
                   nfold=10,type.measure="mse")
```

- Now plot **lasso.cv**.
- What is the size of the model that has min λ and the one that has λ that is within one standard error of the minimum?

```
plot(lasso.cv)
```



- Let's do prediction on the test cases using both values of λ, **lambda.min** and **lambda.1se**, the value of λ that gives the most regularized model such that the cross-validated error is within one standard error of the minimum.
- Save the outputs as **lasso.lambdabest** and **lasso.lambda1se**

```
lasso.lambdabest=predict(lasso.cv,s=lasso.cv$lambda.min,
                                              newx=x[-train,])

lasso.lambda1se=predict(lasso.cv,s=lasso.cv$lambda.1se,
                                              newx=x[-train,])
```

- Compute RMSE of the predictions

```
rmse.lasso.lambdabest <- sqrt(mean((lasso.lambdabest-
                                      y[-train])^2))
rmse.lasso.lambda1se <- sqrt(mean((lasso.lambda1se-
                                      y[-train])^2))

RMSE[5:6,1] <- c(rmse.lasso.lambda1se,rmse.lasso.lambdabest
```

- Model performance across models

```
RMSE
```

```
                         [,1]
rmse.ridge.lambdabig      405
rmse.ridge.lambda4        296
rmse.ridge.lambda0        300
rmse.ridge.lambdabest     292
rmse.lasso.lambda1se      334
rmse.lasso.lambdabest     297
rmse.elnet.lambda1se       NA
rmse.elnet.lambdabest      NA
```

- This is substantially lower than the test set RMSE of the null model and of least squares, however it's larger than the test RMSE of ridge regression with λ chosen by cross-validation

- Let's see which variables are selected by lasso.
- Call the **predict** function on **lasso.cv** and use `s=lasso.cv$lambda.min` and `type="coefficients"`

```
lasso.coef=predict(lasso.cv,s=lasso.cv$lambda.min,
             type="coefficients")[1:20,]
lasso.coef[lasso.coef!=0]
```

```
(Intercept)       AtBat        Hits       HmRun         RBI
    129.174      -1.981       6.307       1.504       1.315
      Years      CHmRun       CRuns      CWalks     LeagueN
     -4.221       0.859       0.718      -0.357      19.539
    PutOuts     Assists      Errors  NewLeagueN
      0.308       0.401      -2.365       1.765
```

- However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 4 of the 19 coefficient estimates are exactly zero.
- So the lasso model with λ chosen by cross-validation contains only 15 variables.