STATS 415 Lab7

Weijing Tang

Feb 22, 2018

1 Today's objectives

1. Randomly split dataset into training and test.
2. Practice how to implement Ridge regression and Lasso with λ chosen by cross-validation.
3. Report the training and test errors for the final models.
4. Compare the model performance among OLS, ridge regression and lasso.

2 Data and necessary packages

We will use the `Hitters` dataset from the `ISLR` package to explore two shrinkage methods: **ridge regression** and **lasso**. These are otherwise known as **penalized regression** methods.

```
library(ISLR)
data(Hitters)
```

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 ~ x syntax. We will now perform ridge regression and the lasso in order to predict Salary on the Hitters data. Before proceeding ensure that the missing values have been removed from the data.

Recall that this dataset has some missing data in the response `Salaray`. We use the `na.omit()` function the clean the dataset.

```
sum(is.na(Hitters))
```

```
## [1] 59
```

```
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

```
Hitters = na.omit(Hitters)
sum(is.na(Hitters))
```

```
## [1] 0
```

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. It also returns the intercept term as the first column, so we need remove the first column by `[,-1]`.

```
library(glmnet)
X = model.matrix(Salary ~ ., Hitters)[, -1]    don't want x to include the intercept
y = Hitters$Salary
```

3 Test-train Split

We begin by splitting the observations into a training set and a test set. We do this by creating a random vector, train, of elements equal to TRUE if the corresponding observation is in the training set, and FALSE otherwise. The vector test has a TRUE if the observation is in the test set, and a FALSE otherwise.

```
set.seed(1)
train=sample(c(TRUE,FALSE), nrow(Hitters),rep=TRUE)
test=(!train)
```

if true, belong to train data, if false, test data

4 Ridge Regression

We first illustrate **ridge regression**, which can be fit using `glmnet()` with `alpha = 0` and seeks to minimize

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2.$$

Question: What is the model if $\lambda = 0$?

more variances included;
lambda=inf no variances included

Notice that the intercept is **not** penalized. Also, note that that ridge regression is **not** scale invariant like the usual unpenalized regression. Thankfully, `glmnet()` takes care of this internally. **It automatically standardizes predictors for fitting, then reports fitted coefficient using the original scale.**

Some commonly used arguments of `glmnet()`: 1. **x**: input matrix 2. **y**: response variable 3. **lambda**: the `glmnet()` function performs ridge regression for an automatically selected range of λ values by default. We can also set a grid of values. Here we choose 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. 4. **alpha**: alpha=0 the ridge penalty and alpha=1 the lasso penalty. 5. **standardize**: the `glmnet()` function standardizes the variables so hat they are on the same scale by default. To turn off this default setting, use the argument `standardize=FALSE`. 6. **thresh**: convergence threshold for optimization method.

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

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 (one for each predictor, plus an intercept) and 100 columns (one for each value of λ). 19+1intercept

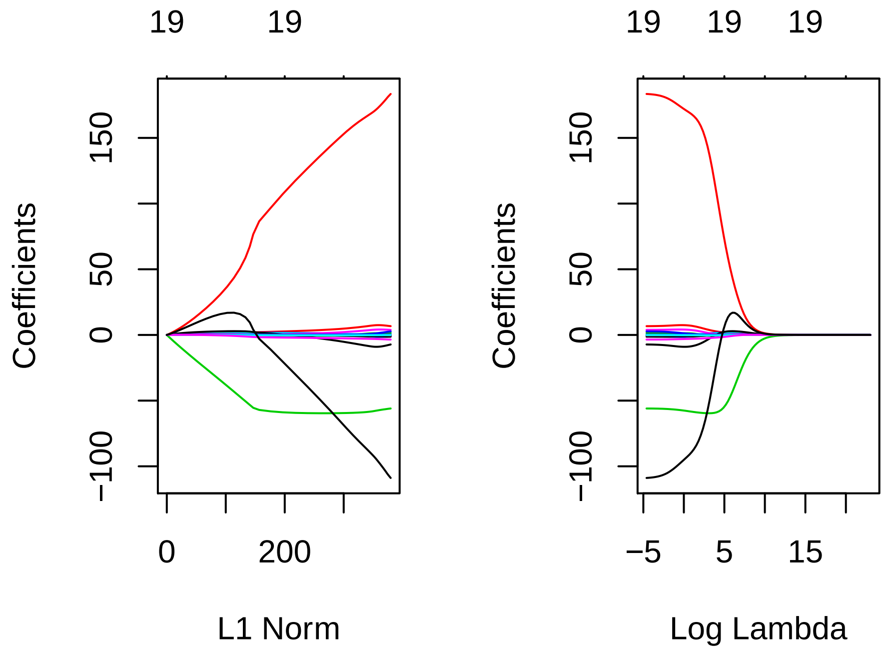
```
dim(coef(ridge.mod))
```

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

Question: What do you expect the coefficient estimates to be if we increase λ? Why?

We can support our conclusion by the following two plots.

```r
par(mfrow = c(1, 2))
plot(ridge.mod)
plot(ridge.mod, xvar = "lambda", label = TRUE)
```



Question: Is there any coefficient is forced to zero?

Here we can again use `predict()` function to make predictions for `newx` and a given λ. We can also obtain the ridge regression coefficients for a new value of λ, say 50:

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

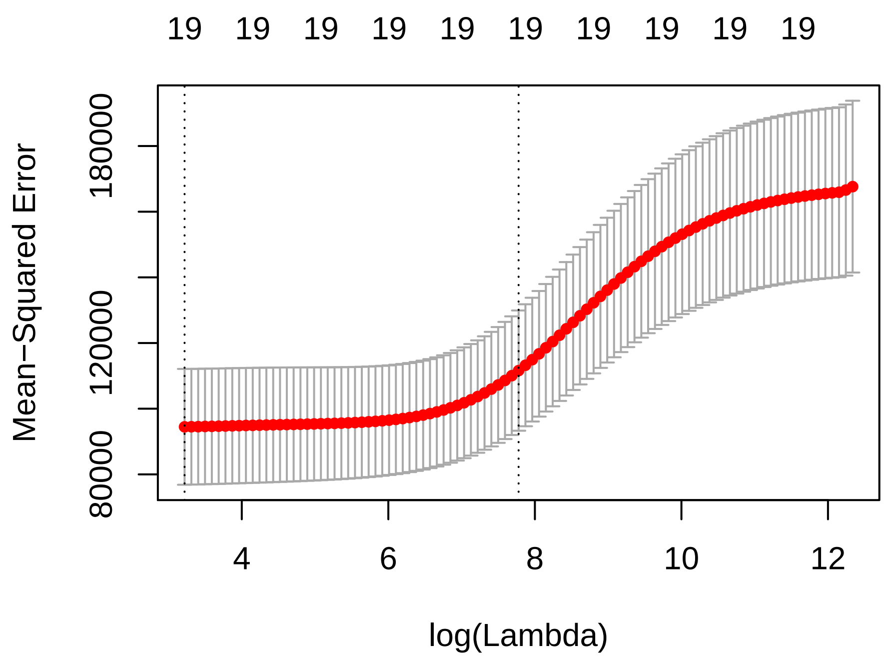
```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## -144.43422527   -0.02343557    2.81521875   -0.54884891    1.59947732
##           RBI         Walks         Years        CAtBat         CHits
##    1.30877723    0.65795601   -0.11157223    0.01693560    0.15314382
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##    0.02074275    0.16694273    0.12683534   -0.25530284  112.48936029
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
##  -59.10973952    0.18318693   -0.26560638   -2.03609898  -25.31027138
```

4.1 Choose *lambda* by cross-valiation

λ is tuning parameter here. We use cross-validation to select a good λ value, which should be based only on training data.

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 so our results will be reproducible, since the choice of the cross-validation folds is random.

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



The plot illustrates the MSE for the λs considered. Two lines are drawn. The first is the λ that gives the smallest MSE. The second is the λ that gives an MSE within one standard error of the smallest.

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

```
## [1] 25.00932
```

Therefore, we see that the value of λ that results in the smallest crossvalidation error is 25.

4.2 Calculating training and test error

```
# training MSE
ridge.pred_train=predict(ridge.mod,s=bestlam,newx=X[train,])
mean((ridge.pred_train-y[train])^2)
```

```
## [1] 67968.27
```

The training MSE is 67968.27.

```
# test MSE
ridge.pred_test=predict(ridge.mod,s=bestlam,newx=X[test,])
mean((ridge.pred_test-y[test])^2)
```

```
## [1] 154665.8
```

The test MSE is 154665.8.

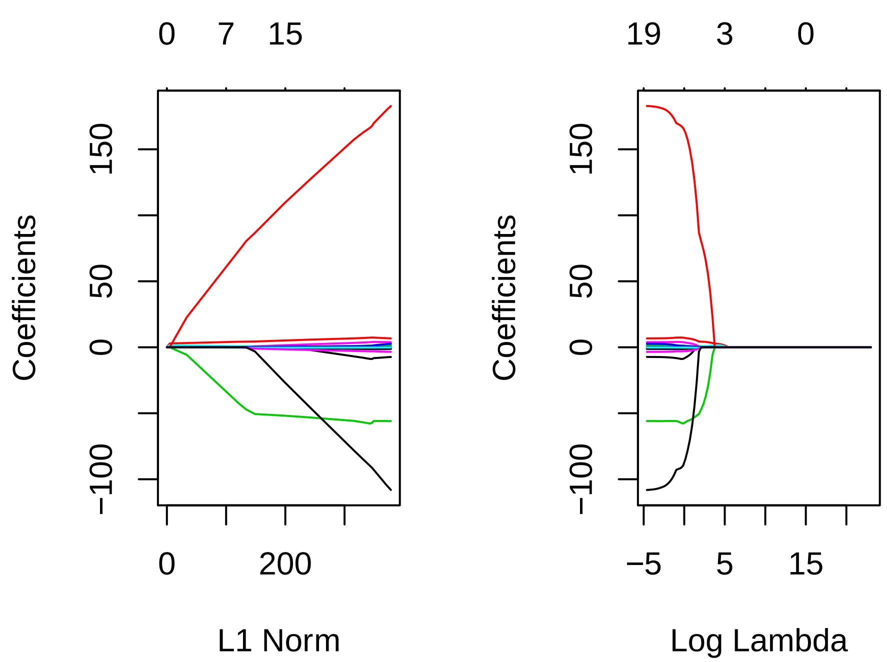
5 Lasso

We now illustrate **lasso**, which can be fit using `glmnet()` with `alpha = 1` and seeks to minimize

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|.$$

Like ridge, lasso is not scale invariant. The only difference between ridge regression and lasso is the penalty. Other than the change `alpha=1`, we proceed just as we did in fitting ridge regression model.

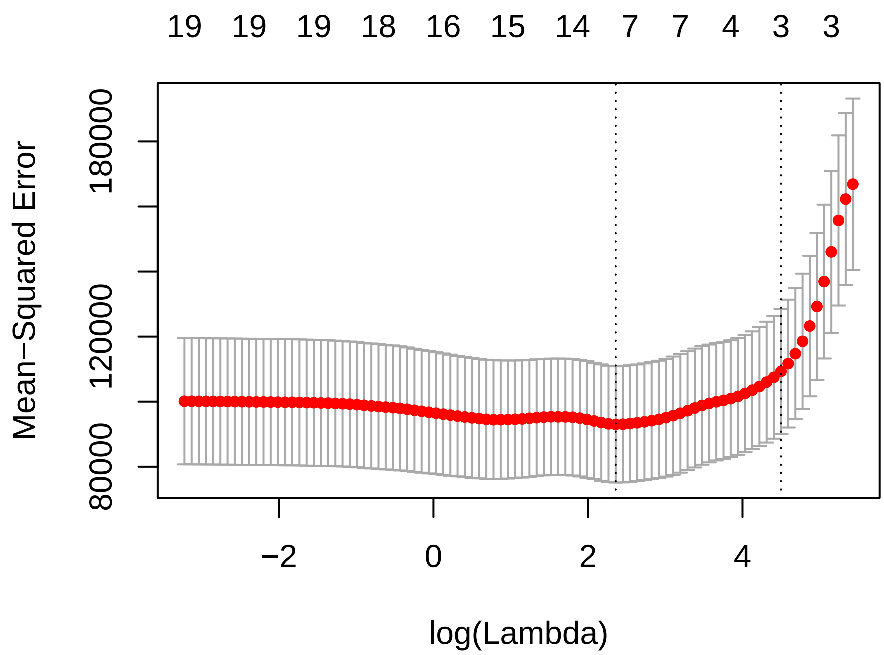
```
lasso.mod=glmnet(X[train,],y[train],alpha=1,lambda=grid)
par(mfrow = c(1, 2))
plot(lasso.mod)
plot(lasso.mod, xvar = "lambda", label = TRUE)
```



Question: Is there any coefficient is forced to zero? What if compared with ridge regression?

We now perform cross-validation and compute the associated test error.

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




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

```
## [1] 10.57705
```

```
# training error
lasso.pred_train=predict(lasso.mod,s=bestlam,newx=X[train,])
mean((lasso.pred_train-y[train])^2)
```

```
## [1] 71678.76
```

```
# test error
lasso.pred_test=predict(lasso.mod,s=bestlam,newx=X[test,])
mean((lasso.pred_test-y[test])^2)
```

```
## [1] 161022.6
```

The test MSE of lasso is larger than that of ridge regression. However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse.

```
lasso.coef=predict(lasso.mod,type="coefficients",s=bestlam)[1:20,]
lasso.coef
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## -1.753112e+02  0.000000e+00  4.207687e+00  0.000000e+00  0.000000e+00
##           RBI         Walks         Years        CAtBat         CHits
```

```
##  7.704335e-01  0.000000e+00  0.000000e+00  0.000000e+00  2.685458e-01
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  0.000000e+00  0.000000e+00  0.000000e+00 -2.526989e-03  7.425729e+01
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -4.324510e+01  1.632754e-01 -3.591127e-01 -1.949470e-02  0.000000e+00
```

Here we see that 10 of 19 coefficient estimates are exactly zero. So the lasso chosen by cross-validation contains only 9 variables.

6 Compare the model performance

We fit the ordinary least square(null model) and calculate traing and test error.

```r
lm.mod = lm(Salary~.,data = Hitters[train,])
# training error
lm.pred_train = predict(lm.mod,Hitters[train,])
mean((y[train]-lm.pred_train)^2)
```

```
## [1] 61988.36
```

```r
# test error
lm.pred_test = predict(lm.mod,Hitters[test,])
mean((y[test]-lm.pred_test)^2)
```

```
## [1] 160105.6
```

Model	Train Error	Test Error
OLS	61988.36	160105.6
Ridge Regression	67968.27	154665.8
Lasso	71678.76	161022.6

Question: We can see that OLS has the lowest training Error. Does it always happen?

Question: Which model should we choose if we want a model to do better at prediction?

Question: Which model should we choose if we prefer a model simple to interpret?

7 Resources and References

1. https://daviddalpiaz.github.io/r4sl/regularization.html#ridge-regression