Penalised Linear Regression Analysis

Johnny Lo

June 2021

Table of Contents

Penalised Linear Regression Analysis

Traditional linear regression analysis, or ordinary least-squares (OLS), are used extensively in a wide variety of scientific fields. However, it often suffers from a number of modelling issues, including multicollinearity and overfitting which can lead to high model variance and low prediction accuracy.

Penalised regression modelling do not have these issues and tend to have more stable results and higher prediction accuracy, which is important in machine learning where accuracy predictions is typically the primary goal. In penalised regression, an penalty term is introduced to the error minimisation process which leads to the "shrinking" of the regression coefficient estimates toward zero. If the shrinking is large enough, some regression coefficients are set to zero exactly. In other words, penalised regression modelling can perform variable selection simultaneously, with coefficient estimation, without compromising its predictive power. Furthermore, penalised regression modelling can be applied even when the the number of features in the model is greater than the number of samples, which is not possible of OLS.

In this workshop, we will consider three(3) penalised regression modelling approaches:

1. Ridge regression
2. LASSO regression
3. Elastic-net regression

Ridge regression

Ridge regression shrinks the regression coefficients of variables that contribute very little to explaining the variability of the outcome variable toward zero. The penalty term responsible for the shrinking of the coefficients is given as the sum of squares of coefficients, which is to be minimised along with the sum of squares of the model residuals. The amount of penalty is controlled by a hyper-parameter $\lambda > 0$. The selection of a good λ value is essential and need to be tuned.

To start, you need to install and load the relevant packages for this workshop.

```
#De-comment to install the packages below
#install.packages(c("tidyverse","timeDate","caret","e1071","glmnet"))
library(tidyverse)  #For ggplot2
library(caret)  #Classification and Regression Training package
library(glmnet)  #For visualisng correlation matrix
library(car)  #For the VIF function
```

We will once again using the CPU data, i.e. *CPU.csv* data file, which contains performance data relating to computer hardware from various vendors and models. The variables in the files are as follows:

(1) Vendor name: 27 (adviser, amdahl, apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec, dg, formation, four-phase, gould, harris, honeywell, hp, ibm, ipl, magnuson, microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens)
(2) Model name: many unique symbols
(3) MYCT: machine cycle time in nanoseconds (integer)
(4) MMIN: minimum main memory in kilobytes (integer)
(5) MMAX: maximum main memory in kilobytes (integer)
(6) CACH: cache memory in kilobytes (integer)
(7) CHMIN: minimum channels in units (integer)
(8) CHMAX: maximum channels in units (integer)
(9) PRP: published relative CPU performance (integer)

To recap, the goal here is to predict relative CPU performance based on its cycle time, memory and etc. The vendors' names and the hardware model, i.e. Variables (1) and (2) are inconsequential in this exercise.

Let us firstly import the data into R studio.

```
CPU <- read.csv("CPU.csv", header=TRUE, stringsAsFactors=FALSE);  #Read in
the data
str(CPU)  #Examine its structure

## 'data.frame':    164 obs. of  9 variables:
##  $ Vendor: chr  "adviser" "amdahl" "amdahl" "amdahl" ...
##  $ Model : chr  "32/60" "470v/7a" "580-5840" "580-5850" ...
##  $ MYCT  : int  125 29 23 23 23 23 400 60 50 350 ...
##  $ MMIN  : int  256 8000 16000 16000 16000 32000 1000 2000 4000 64 ...
##  $ MMAX  : int  6000 32000 32000 32000 64000 64000 3000 8000 16000 64 ...
##  $ CACH  : int  256 32 64 64 64 128 0 65 65 0 ...
##  $ CHMIN : int  16 8 16 16 16 32 1 1 1 1 ...
##  $ CHMAX : int  128 32 32 32 32 64 2 8 8 4 ...
##  $ PRP   : int  198 220 367 489 636 1144 38 92 138 10 ...
```

We will now split the dataset into *training* and *test* sets. The training set is used to build the model, and the test set is used to evaluate its predictive performance. Here, we will use a 75/25 split, i.e. randomly select 75% of the original data to be in the training set, and the remaining 25% form the test set.

```
#Create the training and test datasets

set.seed(1)  #Set the random seed.

# Step 1: Get row numbers for the training data
trainRowNumbers <- createDataPartition(CPU$PRP, #The outcome variable
                                       p=0.75, #proportion of data to form
the training set
                                       list=FALSE  #Don't store the result in
```

```
a list
                                        );

# Step 2: Create the training  dataset, and excluding the 1st 2 columns.
trainData <- CPU[trainRowNumbers,3:9]

# Step 3: Create the test dataset, and excluding the 1st 2 columns.
testData <- CPU[-trainRowNumbers,3:9]
```

To perform penalised regression modelling, we will use the *glmnet(.)* function, concurrently with the *train(.)* function from the **caret** package. We will need to define two parameters for the *train(.)* function.

1) **alpha**: elastic-net mixing parameter
   i) 0 = Ridge regression
   ii) 1 = LASSO regression
   iii) A value between 0 and 1 for elastic-net regression
2) **lambda**: a sequence of lambda values to test with

The *train(.)* function can automatically choose the optimal α (for elastic net) and λ values using cross-validation (CV) for us, or we can define a search range for each of these two parameters.

As we are running ridge regression in this first instance, alpha will be set to 0 here. However, we will create a search range for lambda.

Note that in the lecture slides, the *glmnet(.)* function was used directly to perform penalised regression. This approach performs *k*-fold CV one time only to obtain the optimal λ value whereas multiple rounds of *k*-fold CV can be performed with *train(.)*, which is what we will do here.

```
lambdas <- 10^seq(-3,3,length=200) #A sequence of lambdas

set.seed(1)
mod.ridge <- train(PRP ~., #Formula
                   data = trainData, #Training data
                   method = "glmnet",  #Penalised regression modelling
                   #Set to c("center", "scale") to standardise data
                   preProcess = NULL,
                   #Perform 10-fold CV, 5 times over.
                   trControl = trainControl("repeatedcv",
                                            number = 10,
                                            repeats = 5),
                   tuneGrid = expand.grid(alpha = 0, #Ridge regression
                                          lambda = lambdas)
                  )

#Optimal lambda value
mod.ridge$bestTune
```

```
##     alpha   lambda
## 152     0 35.70786
```

The optimal $\lambda = 35.7$. The regression coefficients of the optimal ridge models are as follows:

```
# Model coefficients
coef(mod.ridge$finalModel, mod.ridge$bestTune$lambda)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept) -16.006679140
## MYCT          0.006240645
## MMIN          0.011708069
## MMAX          0.003665322
## CACH          0.582184188
## CHMIN         6.246722427
## CHMAX         0.009776414
```

Now that we have our optimal ridge regression model, we can now evaluate its predictive performance on the test data. First, we will write a simple function that produces the necessary performance measures.

```
model.pf <- function(actual,predicted)
{
  diff <- actual - predicted
  RMSE <- diff^2 %>% mean %>% sqrt;  #RMSE
  bias <- diff %>% mean  #Bias
  cor <- cor(actual,predicted)  #Correlation
  predR2 <-cor^2  #Prediction R2

  #return as function output
  return(c(RMSE=RMSE, Bias=bias,
           Corr=cor, PredR2=predR2))
}
```

Next, we will predict the PRP at the feature values given by the test set and use the *model.pf(.)* function to summarise the prediction results.

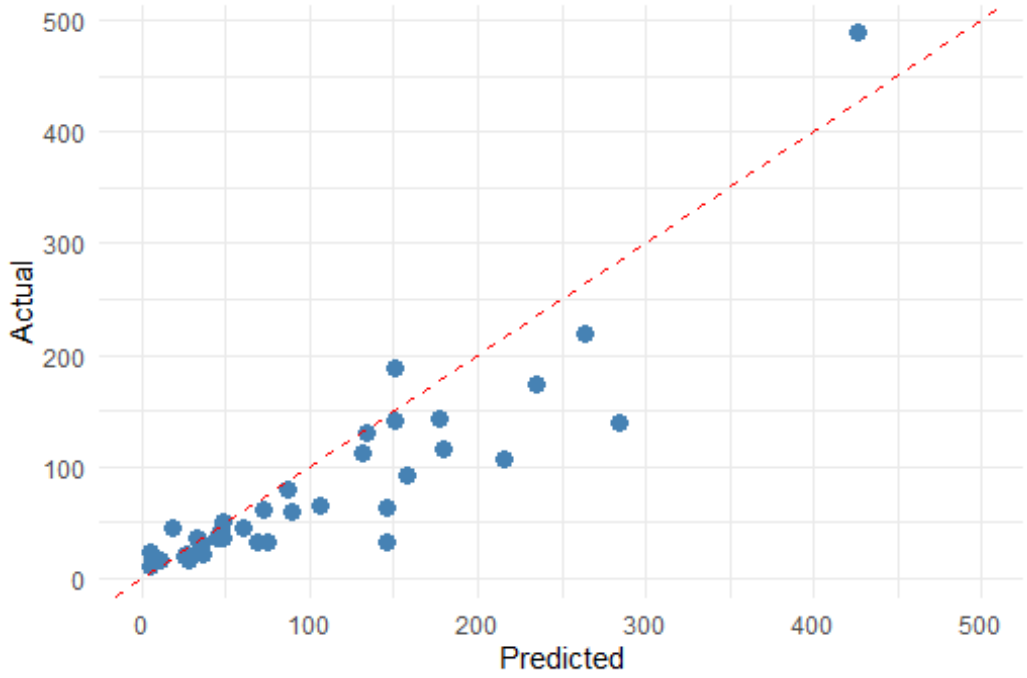
```
pred.ridge <- predict(mod.ridge,testData)  #Predict PRP at the test data
pf.ridge <- model.pf(testData$PRP,pred.ridge); pf.ridge
```

```
##        RMSE        Bias        Corr      PredR2
##  45.2163699 -20.9239905   0.8993482   0.8088271
```

The scatter plot between the actual PRP values from the test set against the ridge regression predicted values is as follows.

```
df <- data.frame(Actual=testData$PRP,Predicted=pred.ridge)

ggplot(df,aes(x=Predicted,y=Actual))+
  geom_point(size=3,colour="steelblue")+
  xlim(0,500)+
  #Reference line given by y=x, i.e. slope=1 and intercept=0
  geom_abline(slope=1,
              intercept=0,
              colour="red",  #Colour of the line
              linetype=2) + #Dotted line
  theme_minimal()
```



As a point of comparison, we will now build an OLS model for predicting PRP and compare its predictive power to ridge regression.

```
mod.ols <- lm(formula=PRP~.,  #PRP as a function of all 6 features
              data=trainData)  #The relevant training data

#Predict PRP with the test data
pred.ols <- predict(mod.ols,  #Model used to make prediction
                    newdata=testData  #Test set
                    )
pf.ols <- model.pf(testData$PRP,pred.ols);

#Compare the results of ridge regression to OLS
rbind(OLS=pf.ols,Ridge=pf.ridge) %>% round(digit=3)
```

```
##         RMSE    Bias  Corr PredR2
## OLS   47.223 -20.194 0.903  0.815
## Ridge 45.216 -20.924 0.899  0.809
```

Exercise: What can you conclude regarding the performance of ridge regression vs OLS?

LASSO Regression

LASSO regression differs to ridge regression in that certain regression coefficient(s) will be set to **exactly zero** as result of the penalty term applied. This also means that LASSO regression can be used as a variable selection technique.

The execution of LASSO regression modelling is exactly the same as ridge regression except the argument alpha is **set to 1**.

```
set.seed(1)
mod.LASSO <- train(PRP ~., #Formula
                   data = trainData, #Training data
                   method = "glmnet",  #Penalised regression modelling
                   #Set preProcess to c("center", "scale") to standardise
data
                   preProcess = NULL,
                   #Perform 10-fold CV, 5 times over.
                   trControl = trainControl("repeatedcv",
                                            number = 10,
                                            repeats = 5),
                   tuneGrid = expand.grid(alpha = 1, #LASSO regression
                                          lambda = lambdas)
                  )

#Optimal lambda value
mod.LASSO$bestTune
```

```
##     alpha   lambda
## 125     1 5.478901
```

The optimal $\lambda = 5.5$. The regression coefficients of the optimal LASSO model are as follows:

```
# Model coefficients
coef(mod.LASSO$finalModel, mod.LASSO$bestTune$lambda)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept) -14.959569742
## MYCT          .
## MMIN          0.015091123
## MMAX          0.003234747
```

```
## CACH          0.551857163
## CHMIN         5.354180679
## CHMAX         .
```

Here we note that both MYCT and CHMAX have been removed from the model, and as a result we have a more simplified model than both OLS and ridge regression. Let us find out what implication this has on the prediction accuracy as compared to the other two models.

```
pred.LASSO <- predict(mod.LASSO,testData)  #Predict PRP at the test data
pf.LASSO <- model.pf(testData$PRP,pred.LASSO); #Performance measures of LASSO

#Tabulate and compare the three models.
mod.comp <- rbind(OLS=pf.ols,Ridge=pf.ridge,LASSO=pf.LASSO) %>% data.frame()
mod.comp %>% round(digit=3)
```

```
##          RMSE    Bias  Corr PredR2
## OLS    47.223 -20.194 0.903  0.815
## Ridge  45.216 -20.924 0.899  0.809
## LASSO  45.375 -20.660 0.905  0.819
```

Even though LASSO is a less complex model, its has outperformed the OLS model, and perform just as well as the ridge regression model in all aspects.

Elastic-Net Regression

Elastic-Net is a compromise between ridge and LASSO regression. In elastic-net regression, we can define a search range for alpha (α), just as we did with lambda. Note that $0 \leq \alpha \leq 1$.

```
alphas <- seq(0.1,0.9,by=0.1); alphas
```

```
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

```
set.seed(1)
mod.elnet <- train(PRP ~., #Formula
                   data = trainData, #Training data
                   method = "glmnet",  #Penalised regression modelling
                   #Set preProcess to c("center", "scale") to standardise
data
                   preProcess = NULL,
                   #Perform 10-fold CV, 5 times over.
                   trControl = trainControl("repeatedcv",
                                            number = 10,
                                            repeats = 5),
                   tuneGrid = expand.grid(alpha = alphas,
                                          lambda = lambdas)
                  )
```

```
#Optimal lambda value
mod.elnet$bestTune
```

```
##     alpha   lambda
## 151   0.1 33.31295
```

The optimal α and λ values are 0.1 and 33.3, respectively. The regression coefficients of the optimal elastic-net model are as follows:

```
# Model coefficients
coef(mod.elnet$finalModel, mod.elnet$bestTune$lambda)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept) -11.939554244
## MYCT          .
## MMIN          0.011820514
## MMAX          0.003545773
## CACH          0.547426551
## CHMIN         6.082844382
## CHMAX         .
```

Just like the LASSO regression model, both MYCT and CHMAX have been omitted from the elastic-net model. Let us see how good the predictions of this model are as compared to the others.

```
pred.elnet <- predict(mod.elnet,testData)  #Predict PRP at the test data
pf.elnet <- model.pf(testData$PRP,pred.elnet); #Performance measures

#Tabulate and compare all 4 models.
mod.comp <- rbind(mod.comp,ElastNet=pf.elnet) %>% data.frame()
mod.comp %>% round(digit=3)
```

```
##            RMSE    Bias  Corr PredR2
## OLS      47.223 -20.194 0.903  0.815
## Ridge    45.216 -20.924 0.899  0.809
## LASSO    45.375 -20.660 0.905  0.819
## ElastNet 44.917 -21.109 0.899  0.809
```

Exercise: Based on the above table, which model would be the appropriate choice as the predictive model for PRP?