Training Linear Regression Model in R

Machine Learning and Applications

Tufan Bostan
12 03 2021

1.Abstract

In this study we will train a linear regression model using R. Here we using “longley” data from AER package. First, we will split “Longley” data into two part which are “Train” and “Test”. Then the “Train” part of data will be summarized using regression methods. After that, using the model we have obtained using "Train" data, predictive values for "Train" and "Test" will be calculated. Finally, using the estimated values obtained, the estimation performance of the model will be interpreted according to their errors.

2.Data Structure

First of all, we have to install “AER” package to access “longley” data (`install.packages("AER")`). After that we need to call the “AER” package to access all of its features. (`library(AER)`) . Now we have “longley” data and whole data are shown below.

```
#install.packages("AER")
library(AER)

longley

##      GNP.deflator     GNP Unemployed Armed.Forces Population Year Employed
## 1947         83.0 234.289      235.6        159.0    107.608 1947   60.323
## 1948         88.5 259.426      232.5        145.6    108.632 1948   61.122
## 1949         88.2 258.054      368.2        161.6    109.773 1949   60.171
## 1950         89.5 284.599      335.1        165.0    110.929 1950   61.187
## 1951         96.2 328.975      209.9        309.9    112.075 1951   63.221
## 1952         98.1 346.999      193.2        359.4    113.270 1952   63.639
## 1953         99.0 365.385      187.0        354.7    115.094 1953   64.989
## 1954        100.0 363.112      357.8        335.0    116.219 1954   63.761
## 1955        101.2 397.469      290.4        304.8    117.388 1955   66.019
## 1956        104.6 419.180      282.2        285.7    118.734 1956   67.857
## 1957        108.4 442.769      293.6        279.8    120.445 1957   68.169
## 1958        110.8 444.546      468.1        263.7    121.950 1958   66.513
## 1959        112.6 482.704      381.3        255.2    123.366 1959   68.655
## 1960        114.2 502.601      393.1        251.4    125.368 1960   69.564
## 1961        115.7 518.173      480.6        257.2    127.852 1961   69.331
## 1962        116.9 554.894      400.7        282.7    130.081 1962   70.551
```

Let’s check the data structure of “longley”. For that we need to install another package which called “dplyr” (`install.packages("dplyr")`). Then we call the package(`library(dplyr)`). Now we can use ” `glimpse()`” function to display data. We have 7 different numeric variables. Here, we are going to use Employed as dependent(predicted,target) variable and the rest of them will be our independent(predictor,input) variables.

```
#install.packages("dplyr")
library(dplyr)

glimpse(longley)
```

```
## Rows: 16
## Columns: 7
## $ GNP.deflator <dbl> 83.0, 88.5, 88.2, 89.5, 96.2, 98.1, 99.0, 100.0, 101.2...
## $ GNP          <dbl> 234.289, 259.426, 258.054, 284.599, 328.975, 346.999, ...
## $ Unemployed   <dbl> 235.6, 232.5, 368.2, 335.1, 209.9, 193.2, 187.0, 357.8...
## $ Armed.Forces <dbl> 159.0, 145.6, 161.6, 165.0, 309.9, 359.4, 354.7, 335.0...
## $ Population   <dbl> 107.608, 108.632, 109.773, 110.929, 112.075, 113.270, ...
## $ Year         <int> 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, ...
## $ Employed     <dbl> 60.323, 61.122, 60.171, 61.187, 63.221, 63.639, 64.989...
```

3.Splitting Data

Using "longley" data we create a sample. This sample has two parts one of them is going to be used for training to model and the other part for test the model. These are assigned to variables named `training_data` and `testing_data`, respectively. Also, how many observations they contain is shown below. `training_data` has 12 observations and `testing_data` 4 observations. After this process we are ready to generate linear regression model.

```
set.seed(2380)
data_splitted <- sample(nrow(longley), nrow(longley) * 0.8)
training_data <- longley[data_splitted,]
testing_data  <- longley[-data_splitted,]
nrow(training_data);nrow(testing_data)

## [1] 12

## [1] 4
```

4.Training LRM

To train model we are going to use "`lm()`" function which is already in R library. Our target variable is Employed and we are going to use all other variables as input. Then by "`summary()`" function, we will display model summary. Here, the regression model assigned to "model" (`model <- lm(Employed~.,data=training_data)`) and using "`summary()`" model summary displayed (`summary(model)`).

```
model <- lm(Employed~.,data=training_data)
summary(model)

##
## Call:
## lm(formula = Employed ~ ., data = training_data)
##
## Residuals:
##     1951     1950     1948     1959     1953     1957     1958     1960
##  0.08915 -0.22513 -0.19860 -0.01639 -0.12780  0.17596 -0.11255  0.06803
##     1949     1962     1947     1961
##  0.01854 -0.22977  0.30967  0.24888
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.281e+03  9.667e+02  -2.360   0.0648 .
## GNP.deflator  9.550e-02  1.029e-01   0.928   0.3961
## GNP          -2.674e-02  3.365e-02  -0.795   0.4628
## Unemployed   -1.728e-02  4.758e-03  -3.631   0.0150 *
## Armed.Forces -8.262e-03  2.587e-03  -3.194   0.0242 *
## Population    8.828e-02  2.482e-01   0.356   0.7366
## Year          1.200e+00  4.981e-01   2.408   0.0610 .
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2735 on 5 degrees of freedom
## Multiple R-squared:  0.9978, Adjusted R-squared:  0.9952
## F-statistic: 384.4 on 6 and 5 DF,  p-value: 1.709e-06
```

Initially, results of model summary illustrate, R-squared and Adj. R-Squared values are approximately 0.99 which means our independent variables highly successful at prediction of dependent variables. Secondly, estimates values represents each independent variable's coefficient in the model. These values give information about which independent variable how much effects on dependent variable. For example, GNP's estimation value is -0.02674, and it has negative effect on Employed that means it decreases predicted Employed value. On the other hand, all coefficients have p-values which describes their significancy. In this result, Intercept and Year variables are significant at %10 significance level. Unemployed an Armed.Forces variables are significant at %5 significance level but rest of the variables are not significant. Lastly, according to p-value of the model we conclude that model is significant.

Now the model is trained and we are ready to obtain predictions according to model which trained using "training_data" data.

```
predicted_train <- predict(model,training_data)
modelEvaluation_training <- data.frame(training_data$Employed,predicted_train)
colnames(modelEvaluation_training) <- c("Actual","Predicted_train")
modelEvaluation_training
```

```
##      Actual Predicted_train
## 1951 63.221        63.13185
## 1950 61.187        61.41213
## 1948 61.122        61.32060
## 1959 68.655        68.67139
## 1953 64.989        65.11680
## 1957 68.169        67.99304
## 1958 66.513        66.62555
## 1960 69.564        69.49597
## 1949 60.171        60.15246
## 1962 70.551        70.78077
## 1947 60.323        60.01333
## 1961 69.331        69.08212
```

Here we have actual values and predicted ones of "training_data". The conclusion here shows that the actual values and predicted values are quite close to each other. This is a situation we want. This means, it can be assumed that our model makes a successful prediction. Now we will compare the predicted values of model by test data.

```
predicted_test <- predict(model,testing_data)
modelEvaluation_test <- data.frame(testing_data$Employed,predicted_test)
colnames(modelEvaluation_test) <- c("Actual","Predicted_train")
modelEvaluation_test
```

```
##      Actual Predicted_test
## 1952 63.639       64.01596
## 1954 63.761       63.78404
## 1955 66.019       65.69655
## 1956 67.857       67.05853
```

We compared the predictions of the model we made using the data of "training_data" before with the actual values of "training_data" and we saw that the results were very close to each other. Now we compare the predictions of the same model with the data we reserved for testing. Here our Actual and Predicted values are still close each other so we can conclude that the model is successful.

5.Measuring Test Performance of Model

In this section we are going to test performance of model. For this, we are going to use Mean Square Error, Root Mean Square Error, Mean Abs. Error and Mean Abs. Per. Error. Each of them gives information about performance of the test own they own. But we are going to calculate all of them to see if there any other conclusion may occur. First, we going to obtain results for train,

```
mse_train <- mean((modelEvaluation_training$Actual - modelEvaluation_training$Pred
icted_train)^2)
mae_train <- mean(abs(modelEvaluation_training$Actual - modelEvaluation_training$P
redicted_train))
rmse_train <- sqrt(mse_train)
mape_train <- mean(abs((modelEvaluation_training$Actual-modelEvaluation_training$P
redicted_train)/modelEvaluation_training$Actual)) * 100
mse_train;rmse_train;mae_train;mape_train

## [1] 0.03115892

## [1] 0.1765189

## [1] 0.151705

## [1] 0.2340278
```

Now, test scores of “training_data” were obtained. Lets calculate for “test_data”,

```
mse_test <- mean((modelEvaluation_test$Actual - modelEvaluation_test$Predicted_tes
t)^2)
mae_test <- mean(abs(modelEvaluation_test$Actual - modelEvaluation_test$Predicted_
test))
rmse_test <- sqrt(mse_test)
mape_test <- mean(abs((modelEvaluation_test$Actual-modelEvaluation_test$Predicted_
test)/modelEvaluation_test$Actual)) * 100
mse_test;rmse_test;mae_test;mape_test

## [1] 0.2210413

## [1] 0.4701503

## [1] 0.3802314

## [1] 0.5734
```

Here, we have all scores for test. Let's make it a table for clear view.

```
performence <- data.frame("Train"=c(mse_train,rmse_train,mae_train,mape_train),
           "Test"=c(mse_test,rmse_test,mae_test,mape_test),
           "Difference"=c(mse_train-mse_test,rmse_train-rmse_test,mae_train-mae_te
st,mape_train-mape_test),
           row.names = c("Mean S. Error", "Root Mean S. Error","Mean Abs. Error","
Mean Abs. Per. Error"))
performence
```

```
##                            Train      Test Difference
## Mean S. Error        0.03115892 0.2210413 -0.1898823
## Root Mean S. Error   0.17651888 0.4701503 -0.2936314
## Mean Abs. Error      0.15170499 0.3802314 -0.2285264
## Mean Abs. Per. Error 0.23402782 0.5734000 -0.3393722
```

The result shows that errors that we obtained for each test and train are close each other. Then we can conclude that our model is good but according to their difference, we might suspect of overfitting problem.