Training a Logistic Regression Model in R

Machine Learning and Applications

Tufan Bostan
20 03 2021

In this study we will train a logistic regression model using R. Here we using “SportsCards” data from AER package. First, we will split “SportsCards” 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, we will talk about model performance.

Data Structure

First of all, we have to install “AER” package to access “SportsCards” data (`install.packages("AER")`). After that we need to call the “AER” package to access all of its features. (`library(AER)`) . Now we have “SporstsCards” data and first 6 observations are listed below.

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

data(SportsCards)
data <- SportsCards
head(data)
```

```
##   good dealer permonth years    income gender        education age trad
e
## 1    B    yes       70    12  [50, 75)   male post-high school  37   ye
s
## 2    B    yes       40     2  [40, 50)   male       4y college  40   ye
s
## 3    B    yes       35    10 [75, 100)   male       4y college  29    n
o
## 4    A    yes       33     3  [40, 50)   male post-high school  42   ye
s
## 5    A    yes       32    10  [30, 40)   male       2y college  25    n
o
## 6    A    yes       30    10  [20, 30)   male  graduate school  24   ye
s
```

Splitting Data

Using “SportsCards”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 118 observations and `testing_data` 30 observations. After this process we are ready to generate regression model.

```
set.seed(2380)
data_splitted <- sample(nrow(data), nrow(data) * 0.8) #Spliting data into
two parts
```

```
training_data <- data[data_splitted,]
testing_data  <- data[-data_splitted,]
nrow(training_data);nrow(testing_data)
```

```
## [1] 118
```

```
## [1] 30
```

```
summary(SportsCards$trade)
```

```
##  no yes
##  98  50
```

Also, according to summary of trade we can say that, our data has more “no” than “yes”. If these two values are close to each other, it may reduce the chance of encountering overfit and underfit problems. If the amount of “no” or “yes” is too much compared to the other, the more one can be learned well by the model, and the other observation may cause us to get erroneous results because it cannot be learned well with the model.

Training Model

To train model we are going to use ”`glm()`” function which is already in R library. Our target variable is “trade” 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 <- glm(trade ~.,data = training_data, family = "binomial")`) and using “`summary()`” model summary displayed (`summary(model)`). Here we used `family = "binomial”` because we have two levels in our response variable.

```
model <- glm(trade ~.,data = training_data, family = "binomial")
summary(model)
```

```
##
## Call:
## glm(formula = trade ~ ., family = "binomial", data = training_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5350  -0.8103  -0.5008   0.9535   2.2264
##
## Coefficients:
##                             Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -0.06663    0.99476  -0.067   0.9466
## goodB                      -0.01795    0.48203  -0.037   0.9703
## dealeryes                   0.96727    0.56855   1.701   0.0889 .
## permonth                    0.03623    0.02605   1.391   0.1643
## years                      -0.04037    0.03609  -1.118   0.2634
## income[10, 20)             -0.05584    1.28089  -0.044   0.9652
## income[20, 30)             -0.88192    1.21583  -0.725   0.4682
## income[30, 40)             -1.99093    1.19748  -1.663   0.0964 .
## income[40, 50)             -0.94540    1.09468  -0.864   0.3878
## income[50, 75)             -1.07051    1.14842  -0.932   0.3513
## income[75, 100)            -1.89027    1.30947  -1.444   0.1489
## income[100, Inf)          -16.11681 1455.39822  -0.011   0.9912
```

```
## genderfemale                -1.14480    0.83974  -1.363   0.1728
## educationhigh school        -1.84770    1.12029  -1.649   0.0991 .
## education2y college          0.27146    1.37991   0.197   0.8440
## educationpost-high school   -0.65648    1.30413  -0.503   0.6147
## education4y college         -0.18975    1.24247  -0.153   0.8786
## educationgraduate school    -1.00073    1.36954  -0.731   0.4650
## age                          0.01350    0.02728   0.495   0.6207
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 146.77  on 117  degrees of freedom
## Residual deviance: 123.12  on  99  degrees of freedom
## AIC: 161.12
##
## Number of Fisher Scoring iterations: 14
```

First of all, the hypothesis regarding this result is $H_0: \beta_i = 0,\ H_1: \beta_i \neq 0,\ i = 0,1,\ldots$. If H0 is not rejected, it means that the coefficient of that variable is not significant. In this case, it means that this variable does not make a significant difference in the model. This decision is made by looking at the p values in the output. These values denoted under `Pr(>|z|)`. Now, all p-values are greater than 0.05 which means any of them is not significant at 0.05 significant level on this model. For all coefficients, H_0 could not rejected. Input variables does not have a significant statistical effect on the response variable. "Estimates" of these coefficients represent that, effectivity of input variables on response variable. For example, `goodB` has `-0.01795` Estimate value this illustrates that, if `goodB` increases one unit, our response variable decreases 0.01795 times of that unit.

Now, although not a good model, the model is trained and we are ready to obtain predictions according to model which trained using "training_data" data.

```
predicted_probs_train <- predict(model,type = "response")
predicted_class_train <- ifelse(predicted_probs_train > 0.5, "yes", "no")
mean(predicted_class_train == training_data$trade)
```

```
## [1] 0.7457627
```

Here we predicted the "trade" by using "training_data". As a result of this, we get estimation values which are between 1 and 0. We will round the ones greater than 0.5 to 1, and the smaller ones to 0. But our response variable is not in that form. We have "no" and "yes" so, instead of rounding to 1 and 0, we will rename them to "no" or "yes". Here, estimation values renamed as the ones greater than 0.5 as "yes", and the smaller ones as "no". At the and we calculated the accuracy ratio. This ratio represents, model's performance on "trainig_data". The model classifies the "training_data" approximately 75% correctly.

```
predicted_probs_test <- predict(model,testing_data,type = "response")
predicted_class_test <- ifelse(predicted_probs_test > 0.5, "yes", "no")
mean(predicted_class_test == testing_data$trade)
```

```
## [1] 0.6
```

We did the same process that we did whit "training_data" before to testing data. The accuracy ratio that we obtained using the "training_data" is greater than testing_data. We might suspect of

overfitting problem. Here, the higher this ratio means, the better estimation result we get. The model classifies the "testing_data" 60% correctly. Although this value is not very high, it would be wrong to say that it failed. For final decision, we will construct confusion matrix,

```
table(predicted=predicted_class_test,actual=testing_data$trade)
```

```
##          actual
## predicted no yes
##       no  14   9
##       yes  3   4
```

Finally, we have confusion matrix. We can conclude that, the model good at classification to "no".

Accuracy = (14+4)/30 = 18/30 = 0.6

Sensitivity(?) = 4/13 = 0.307 (True Positive)

Specificity(?) = 14/17 = 0.823 (True Negative)

We have 14 true classifications and only 3 wrong classifications which is pretty accurate. On the other hand, the model could not classify "yes" well. There are 4 correct classifications and 9 false classifications. We can say that the model fails to classify "yes". We mentioned earlier that input variables are not significant, as we observed in the model summary, that might be caused this fault. Also, we had more "no" observations than "yes" this is also might be the reason of this bad result. To fix this problem, we might remove some features or training whit more data etc.