Heart attack risk classification with logistic regression

Logistic Regression Model for Heart Attack Prediction

In this study, we will examine the logistic regression model for the heart attack data set, which will classify the risk of heart attack according to various characteristics of individuals (age, sex, Chest pain..).

- Describe data set
- Training logistic regression model
- The performance of trained model
- Checking imbalance problem status

Packages

We loaded the *caret* library for the confusion matrix and included it in the project. We also add the ROSE library to implement the over and under sampling method.

```
#install.packages("caret")
#install.packages("ROSE")

library(caret)
```

```
library(ROSE)
```

Heart Attack Data set

The data set consists of a class of labels that show that the risk of heart attack is high or low, using 13 different characteristics of individuals. We download the data set we used in the study from Kaggle Heart Attack Dataset with .csv extension.

```
HeartData<- read.csv("heart.csv")
```

We are clearing the NA data in the dataset.

```
HeartData<-na.exclude(HeartData)
```

Column attributes

The column information of our imported dataset is as follows.

- Age : Age of the patient
- Sex : Sex of the patient
- exang: exercise induced angina (1 = yes; 0 = no)
- ca: number of major vessels (0-3)
- cp : Chest Pain type chest pain type
  - Value 1: typical angina
  - Value 2: atypical angina
  - Value 3: non-anginal pain
  - Value 4: asymptomatic
- trtbps : resting blood pressure (in mm Hg)
- chol : cholestoral in mg/dl fetched via BMI sensor
- fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- rest_ecg : resting electrocardiographic results
  - Value 0: normal
  - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
  - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
- thalach : maximum heart rate achieved
- target : 0= less chance of heart attack 1= more chance of heart attack

We can look at the data set with the **str()** function. The function returns values containing the following information:

- 303 observation
- 14 variables (features)

  The types and some values of the features are shown in the figure below.

```
str(HeartData)
```

```
'data.frame':   303 obs. of  14 variables:
 $ age     : int  63 37 41 56 57 57 56 44 52 57 ...
 $ sex     : int  1 1 0 1 0 1 0 1 1 1 ...
 $ cp      : int  3 2 1 1 0 0 1 1 2 2 ...
 $ trtbps  : int  145 130 130 120 120 140 140 120 172 150 ...
 $ chol    : int  233 250 204 236 354 192 294 263 199 168 ...
 $ fbs     : int  1 0 0 0 0 0 0 0 1 0 ...
 $ restecg : int  0 1 0 1 1 1 0 1 1 1 ...
 $ thalachh: int  150 187 172 178 163 148 153 173 162 174 ...
 $ exng    : int  0 0 0 0 1 0 0 0 0 0 ...
 $ oldpeak : num  2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
 $ slp     : int  0 0 2 2 2 1 1 2 2 2 ...
 $ caa     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ thall   : int  1 2 2 2 2 1 2 3 3 2 ...
 $ output  : int  1 1 1 1 1 1 1 1 1 1 ...
```

Task: Predict Heart Attack

In this study, we try to estimate the probability of individuals having a heart attack using the Heart attack data set to train the logistic regression model.

Step 1- Splitting the data set

We use the *sample()* function to split 80% of the data into the train and 20% into the test set and set the *seed()* to keep the same values for every future run.

```
set.seed(123)

index <-sample(1:nrow(HeartData),round(nrow(HeartData))*0.80)
traindata <-HeartData[index,]
testdata <-HeartData[-index,]
```

Here, randomly selected indexes are assigned to train and test sets.

Step 2- Train a logistic regression

We use the "glm()" function to train our logistic regression model. This function needs 3 parameters. The first is the formula of the model, the second is the data set used to train the model, and finally the family distribution of the target feature. We set it to "*binomial*" because we will produce binary outputs in family distribution logistic regression models. In the study, we defined the model formula as $y \sim .$. Here "y" is the target variable representing the *output* property to be estimated in our data set. In addition, with this definition, we trained our model according to all variables.

```
Heartmodel<-glm(output~.,data=traindata,family = "binomial")
```

With the *summary()* function, we can see detailed information about our logistic regression model.

```
summary(Heartmodel)
```

```
Call:
glm(formula = output ~ ., family = "binomial", data = traindata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6127  -0.4311   0.1795   0.5835   2.4205

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  4.555347   2.794683   1.630  0.10310
age         -0.013283   0.025289  -0.525  0.59941
sex         -1.540661   0.509861  -3.022  0.00251 **
cp           0.769378   0.195309   3.939 8.17e-05 ***
trtbps      -0.017916   0.011003  -1.628  0.10348
chol        -0.003182   0.004177  -0.762  0.44617
fbs          0.132748   0.566248   0.234  0.81465
restecg      0.658516   0.389009   1.693  0.09049 .
thalachh     0.019761   0.011439   1.727  0.08408 .
exng        -1.077694   0.448609  -2.402  0.01629 *
oldpeak     -0.678520   0.240961  -2.816  0.00486 **
slp          0.263729   0.411859   0.640  0.52195
caa         -0.646435   0.201346  -3.211  0.00132 **
thall       -1.015510   0.330194  -3.075  0.00210 **
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 333.48  on 241  degrees of freedom
Residual deviance: 173.07  on 228  degrees of freedom
AIC: 201.07

Number of Fisher Scoring iterations: 6
```

The Performance of Trained Model

Checking the performance of the model with the test data set is necessary for the model to have good generalization. For this, we first try to estimate the *output* values in the test set. When calculating this, we need to exclude the *output* property from the test set.

```r
predicted_hmodel <- predict(Heartmodel, testdata[,-14], type = "response")
head(predicted_hmodel)
```

```
        2         3        12        15        19        28
0.7079596 0.9397678 0.9823383 0.9748455 0.6193038 0.9260086
```

predicted_hmodel is a vector that calculates the probability of an individual's heart attack. We convert these values into classes.

```r
predicted_output <- ifelse(predicted_hmodel > 0.5, 1, 0)
head(predicted_output)
```

```
 2  3 12 15 19 28
 1  1  1  1  1  1
```

To calculate metrics based on the confusion matrix, let's assume that individuals with an output value of 1 (high risk of heart attack) are the positive class and individuals with 0 (low risk of heart attack) are the negative class.

```r
TP <- sum(predicted_output[which(testdata$output == 1)] == 1)
FP <- sum(predicted_output[which(testdata$output == 1)] == 0)
TN <- sum(predicted_output[which(testdata$output == 0)] == 0)
FN <- sum(predicted_output[which(testdata$output == 0)] == 1)
```

After finding the number of correctly or incorrectly predicted values, we calculate model performance metrics.

```
recall <- TP / (TP + FN)
specificity <- TN / (TN + FP)
precision <- TP / (TP + FP)
accuracy <- (TN + TP) / (TP + FP + TN + FN)
```

```
recall
```

```
[1] 0.7948718
```

```
specificity
```

```
[1] 0.9090909
```

```
precision
```

```
[1] 0.9393939
```

```
accuracy
```

```
[1] 0.8360656
```

We can also use the confusion matrix to see the performance metrics (accuracy.precision,recall and others) in detail and to see the correct and incorrectly classified values.

```
confusionMatrix(table(ifelse(testdata$output == 1, "1", "0"),
                      predicted_output),
                positive = "1")
```

```
Confusion Matrix and Statistics

   predicted_output
     0  1
  0 20  8
  1  2 31
```

```
               Accuracy : 0.8361
                 95% CI : (0.7191, 0.9185)
    No Information Rate : 0.6393
    P-Value [Acc > NIR] : 0.000614

                  Kappa : 0.6645

 Mcnemar's Test P-Value : 0.113846

            Sensitivity : 0.7949
            Specificity : 0.9091
         Pos Pred Value : 0.9394
         Neg Pred Value : 0.7143
             Prevalence : 0.6393
         Detection Rate : 0.5082
   Detection Prevalence : 0.5410
      Balanced Accuracy : 0.8520

       'Positive' Class : 1
```

Checking Imbalance Problem Status

In general, the most used metric according to the performance metric values, the accuracy value is **0.83**, the precision is **0.93**, the specificity is **0.83** and the recall value are **0.79**. We can check this situation for balance of classes in target property.

```
table(traindata$output) / dim(traindata)[1]
```

```
        0         1
0.4545455 0.5454545
```

It is seen that the classes of the target feature are almost balanced.

Over sampling method

We look at the distribution of classes in the Train data.

```
table(traindata$output)
```

```
  0   1
110 132
```

We apply over sampling method to the data. Since the top data number of the class is 132, twice as much data is produced.

```
data_balanced_over <- ovun.sample(output~ ., data = traindata, method = "over",N = 264)$da
table(data_balanced_over$output)
```

```
  0   1
132 132
```

We train our model again with new data. We test the test data with the trained model.

```
Heartmodel_over<-glm(output~.,data=data_balanced_over,family = "binomial")
```

```
predicted_hmodel_over <- predict(Heartmodel_over, testdata[,-14], type = "response")
head(predicted_hmodel_over)
```

```
        2         3        12        15        19        28
0.6186246 0.9679032 0.9965988 0.9783344 0.7530356 0.9801901
```

```
predicted_output_over <- ifelse(predicted_hmodel_over > 0.5, 1, 0)
head(predicted_output_over)
```

```
 2  3 12 15 19 28
 1  1  1  1  1  1
```

We calculate the performance values of the model trained according to over sampling.

```
confusionMatrix(table(ifelse(testdata$output == 1, "1", "0"),
                      predicted_output_over),
                positive = "1")
```

```
Confusion Matrix and Statistics

   predicted_output_over
     0  1
  0 20  8
  1  2 31

               Accuracy : 0.8361
                 95% CI : (0.7191, 0.9185)
    No Information Rate : 0.6393
    P-Value [Acc > NIR] : 0.000614

                  Kappa : 0.6645

 Mcnemar's Test P-Value : 0.113846

            Sensitivity : 0.7949
            Specificity : 0.9091
         Pos Pred Value : 0.9394
         Neg Pred Value : 0.7143
             Prevalence : 0.6393
         Detection Rate : 0.5082
   Detection Prevalence : 0.5410
      Balanced Accuracy : 0.8520

       'Positive' Class : 1
```

Under samplimg method

We apply the under-sampling method to the data. Since the number of metadata of the class is 110, twice as much data is produced.

```
data_balanced_under <- ovun.sample(output ~ ., data = traindata, method = "under", N = 220

table(data_balanced_under$output)
```

```
  0   1
110 110
```

We train our under-sampling model again with new data. We test the test data with the under-sampling trained model.

```
Heartmodel_under<-glm(output~.,data=data_balanced_under,family = "binomial")
```

```
predicted_hmodel_under <- predict(Heartmodel_under, testdata[,-14], type = "response")
head(predicted_hmodel_under)
```

```
        2         3        12        15        19        28
0.7195084 0.9449831 0.9807693 0.9752567 0.5342878 0.8942059
```

```
predicted_output_under <- ifelse(predicted_hmodel_under > 0.5, 1, 0)
head(predicted_output_under)
```

```
 2  3 12 15 19 28
 1  1  1  1  1  1
```

We calculate the performance values of the model trained according to under-sampling.

```
confusionMatrix(table(ifelse(testdata$output == 1, "1", "0"),
                      predicted_output_under),
                positive = "1")
```

```
Confusion Matrix and Statistics

   predicted_output_under
     0  1
  0 20  8
  1  3 30

               Accuracy : 0.8197
                 95% CI : (0.7002, 0.9064)
    No Information Rate : 0.623
    P-Value [Acc > NIR] : 0.0007305

                  Kappa : 0.6319

 Mcnemar's Test P-Value : 0.2278000
```

```
            Sensitivity : 0.7895
            Specificity : 0.8696
         Pos Pred Value : 0.9091
         Neg Pred Value : 0.7143
             Prevalence : 0.6230
         Detection Rate : 0.4918
   Detection Prevalence : 0.5410
      Balanced Accuracy : 0.8295

       'Positive' Class : 1
```

According to the models trained and tested with unbalanced, over-sampled and under-sampled train data, the accuracy value was found to be **0.83** for unbalanced data, **0.83** for over-sampling and **0.81** for under-sampling. From here, if the model training is done with a large number of balanced data, it will affect the model performance well.