Class

BQOM 2578 | Data Mining

Theresa Wohlever

2025-01-01

Table of contents

Executive Summary

Loading packages

Lets start by calling some libraries that are useful for Logistic Regressions.

- caTools for splitting the data in a smart way.
- ROCR for creating ROC and AUC curves.
- caret for Machine Learning in general.

Importing data

We are using a *loans* dataset from Taiwan, Credit_data.csv, which is available on Canvas. Download it to your working folder and use getwd() and setwd() if you need to change your working directory.

Here is information for some of the key variables:

- Limit: Amount of given credit, in 1,000 New Taiwan Dollars (1 USD is approx. 30 NTD, checked in Sept, 2025)
- Gender: 1 = male; 2 = female;
- Marital status: 1 = married; 2 = single/other;
- Late i: whether the person was late i months ago;
- Default: dependent variable (0/1); if the client defaulted (1) or not (0)

Note how gender and Marital Status have 1 and 2 rather than 0 and 1 as we prefer for dummies. We can let R handle this for us.

```
table(loans$Default)
```

```
    0     1
23045  6610
```

```
summary(loans$Default)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.2229  0.0000  1.0000
```

```
paste("The proportion of customers defaulting is: ",round(mean(loans$Default),2))
```

```
[1] "The proportion of customers defaulting is:  0.22"
```

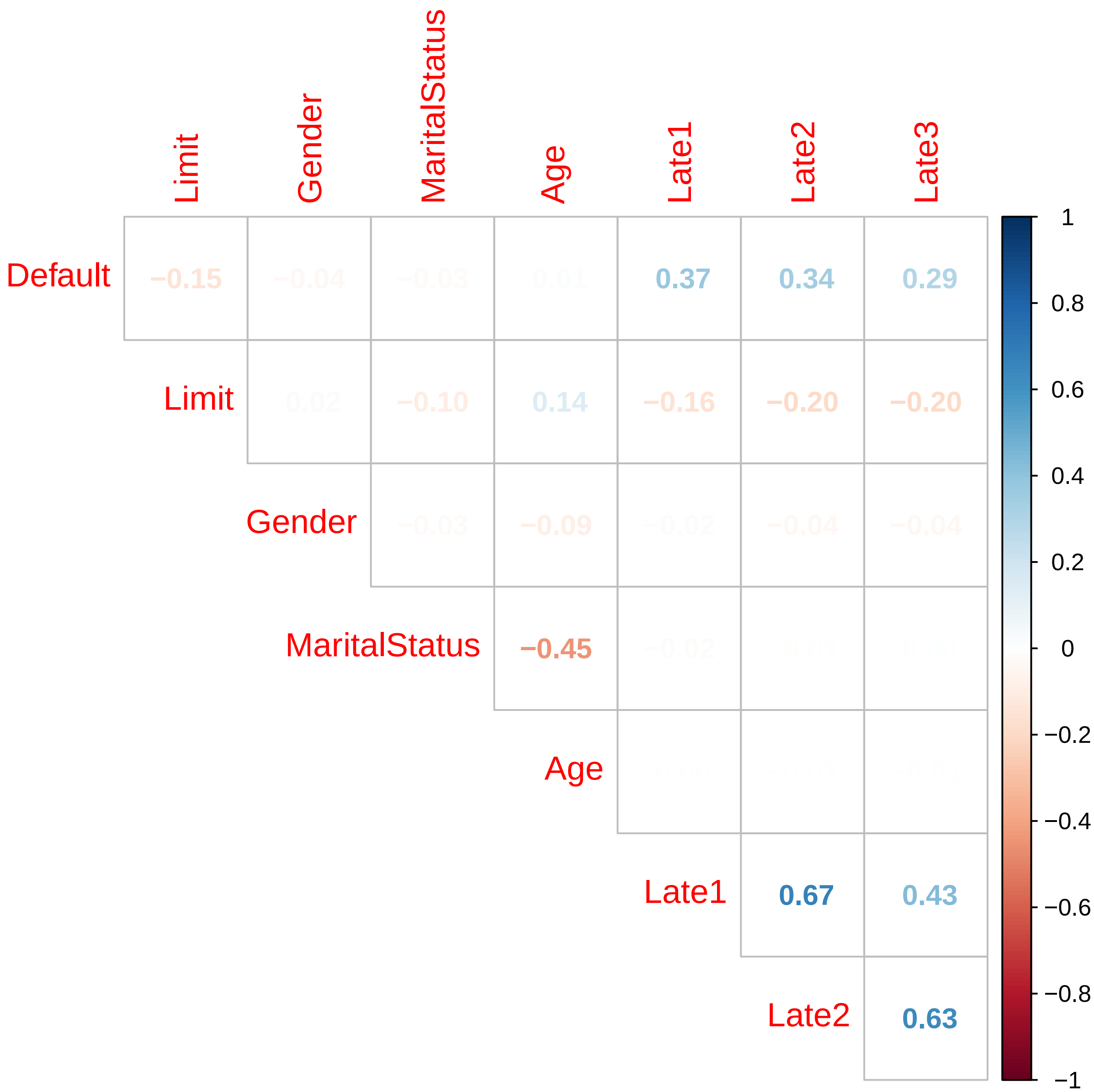
Preliminary Analysis

Let's begin with evaluating the Correlation Matrix

```
loans<-loans%>%relocate(Default)  # moves the variable "Default" to the first column (left
head(loans)
```

```
  Default Limit Gender MaritalStatus Age Late1 Late2 Late3
1       1    20      2             1  24     1     1     0
2       1   120      2             2  26     0     1     0
3       0    90      2             2  34     0     0     0
4       0    50      2             1  37     0     0     0
5       0    50      1             1  57     0     0     0
6       0    50      1             2  37     0     0     0
```

```r
cormat <- round(cor(loans),2)
corrplot(cormat, method="number", type="upper",diag=FALSE, tl.cex=1.2)
```



Logistic Regression Models

Formula is very simple: glm(y ~ X, family="binomial").

For variables that we want to treat as factors (categorical variables) we use as.factor. R will change it to a dummy, taking the lowest value as 0.

Specifically, after using as.factor(Gender), 1 will become 0 and 2 will become 1.

```r
logreg0 = glm(Default ~ Limit, data=loans, family="binomial")
summary(logreg0)
```

```
Call:
glm(formula = Default ~ Limit, family = "binomial", data = loans)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)    
(Intercept) -0.7458015  0.0224716  -33.19   <2e-16 ***
Limit       -0.0033003  0.0001265  -26.09   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 31467  on 29654  degrees of freedom
Residual deviance: 30696  on 29653  degrees of freedom
AIC: 30700

Number of Fisher Scoring iterations: 4
```

```r
logreg = glm(Default ~ Limit + as.factor(Gender) + as.factor(MaritalStatus)
             + Age + Late1 + Late2 + Late3, data=loans, family="binomial")
summary(logreg)
```

```
Call:
glm(formula = Default ~ Limit + as.factor(Gender) + as.factor(MaritalStatus) + 
    Age + Late1 + Late2 + Late3, family = "binomial", data = loans)

Coefficients:
                            Estimate Std. Error z value Pr(>|z|)    
(Intercept)               -1.5306414  0.0838129 -18.263  < 2e-16 ***
Limit                     -0.0019926  0.0001346 -14.800  < 2e-16 ***
as.factor(Gender)2        -0.1372312  0.0313593  -4.376 1.21e-05 ***
as.factor(MaritalStatus)2 -0.1661281  0.0346078  -4.800 1.58e-06 ***
Age                        0.0041690  0.0018393   2.267   0.0234 *  
Late1                      1.3578076  0.0412525  32.915  < 2e-16 ***
Late2                      0.2976810  0.0550675   5.406 6.45e-08 ***
Late3                      0.7161429  0.0472809  15.147  < 2e-16 ***
```

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

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 31467  on 29654  degrees of freedom
Residual deviance: 26975  on 29647  degrees of freedom
AIC: 26991

Number of Fisher Scoring iterations: 4
```

Let's examine the coefficients β and exp(β):

```
coef(logreg)
```

```
              (Intercept)                     Limit        as.factor(Gender)2
             -1.530641432              -0.001992608              -0.137231191
as.factor(MaritalStatus)2                       Age                     Late1
             -0.166128052               0.004168986               1.357807557
                    Late2                     Late3
              0.297681026               0.716142876
```

```
exp(coef(logreg))
```

```
              (Intercept)                     Limit        as.factor(Gender)2
                0.2163968                 0.9980094                 0.8717687
as.factor(MaritalStatus)2                       Age                     Late1
                0.8469378                 1.0041777                 3.8876605
                    Late2                     Late3
                1.3467321                 2.0465243
```

```
coeftable<-data.frame(col1=coef(logreg),col2=exp(coef(logreg)))
colnames(coeftable)<-c('Coefficient (log-odds)','e^coefficient (odds)')

coeftable
```

```
                          Coefficient (log-odds) e^coefficient (odds)
(Intercept)                         -1.530641432            0.2163968
Limit                               -0.001992608            0.9980094
as.factor(Gender)2                  -0.137231191            0.8717687
```

```
as.factor(MaritalStatus)2          -0.166128052          0.8469378
Age                                 0.004168986          1.0041777
Late1                               1.357807557          3.8876605
Late2                               0.297681026          1.3467321
Late3                               0.716142876          2.0465243
```

Confusion Matrix

What we get from the logistic regression are predicted probabilities. What we need to convert them into classification decisions is a threshold or cutoff.

```
loans$PredProbs1<-predict(logreg, newdata=loans, type="response")
# type="response" gives the probability, otherwise the output would be log odds.
summary(loans$PredProbs1)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.03034 0.11892 0.14713 0.22290 0.25709 0.74341
```

```
#We then transform that prediction into either 1 (True) or 0 (False) using a cutoff point
cutoff<-0.25
#Try different cutoff points:
#cutoff<-0.15   # Note the Probs don't change, just the classification
#cutoff<-0.05

loans$PredDefault1<-ifelse(loans$PredProbs1>=cutoff,1,0)
summary(loans$PredDefault1)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.2547  1.0000  1.0000
```

```
paste("For a cutoff point of",cutoff, "the proportion of customers classified as defaulting
```

```
[1] "For a cutoff point of 0.25 the proportion of customers classified as defaulting is 0.25"
```

```
head(loans)
```

```
  Default Limit Gender MaritalStatus Age Late1 Late2 Late3 PredProbs1
1       1    20      2             1  24     1     1     0  0.5119526
2       1   120      2             2  26     0     1     0  0.1588194
3       0    90      2             2  34     0     0     0  0.1333579
4       0    50      2             1  37     0     0     0  0.1661378
5       0    50      1             1  57     0     0     0  0.1989867
6       0    50      1             2  37     0     0     0  0.1621731
  PredDefault1
1            1
2            0
3            0
4            0
5            0
6            0
```

```r
tail(loans)
```

```
      Default Limit Gender MaritalStatus Age Late1 Late2 Late3 PredProbs1
29650       1    80      1             2  34     1     1     1  0.6586309
29651       0   220      1             1  39     0     0     0  0.1410708
29652       0   150      1             2  43     0     0     0  0.1398671
29653       1    30      1             2  37     1     1     1  0.6833773
29654       1    80      1             1  41     1     0     0  0.4597587
29655       1    50      1             1  46     0     0     0  0.1917780
      PredDefault1
29650            1
29651            0
29652            0
29653            1
29654            1
29655            0
```

Let's see what is the combination of Predicted vs Observed Default values (Try with different cutoffs above.)

```r
paste("Observed Default values:")
```

```
[1] "Observed Default values:"
```

```r
table(loans$Default)
```

```
    0     1 
23045  6610 
```

```r
paste("Predicted Default values:")
```

```
[1] "Predicted Default values:"
```

```r
table(loans$PredDefault1)
```

```
    0     1 
22101  7554 
```

```r
paste("Observed by predicted values (confusion matrix) for cutoff of", cutoff, "is:")
```

```
[1] "Observed by predicted values (confusion matrix) for cutoff of 0.25 is:"
```

```r
#first variable is rows, second variable is columns

ConfMatrix<-table(loans$Default,loans$PredDefault1)  # table() with two variables will cros
rownames(ConfMatrix)<-c("Obs False", "Obs True")
colnames(ConfMatrix)<-c("Pred False", "Pred True")
ConfMatrix
```

```
            Pred False Pred True
  Obs False      19198      3847
  Obs True        2903      3707
```

The below function does that. Easier to copy and paste in the future. Use it to test multiple cutoff points.

```r
#defining a new function called ConfMatrix with three inputs:  Actual, Predicted and Cutoff
ConfMatrix_func<-function(actual_value,predicted_prob,cutoff){
  predicted_value<-ifelse(predicted_prob>=cutoff,1,0)  # Classify by evaluating probability
  ConfMatrix<-table(actual_value,predicted_value)
  rownames(ConfMatrix)<-c("Obs False", "Obs True")
  colnames(ConfMatrix)<-c("Pred False", "Pred True")

  print(paste("For a cutoff of", cutoff,":"))
  print("Actual values in the test dataset")
  print(table(actual_value))
  print("Predicted values in the test dataset")
  print(table(predicted_value))

  return(ConfMatrix)
}
```

Let's try it out:

```r
ConfMatrix_func(loans$Default,loans$PredProbs1,0.25)
```

```
[1] "For a cutoff of 0.25 :"
[1] "Actual values in the test dataset"
actual_value
    0     1
23045  6610
[1] "Predicted values in the test dataset"
predicted_value
    0     1
22101  7554
```

```
            predicted_value
actual_value Pred False Pred True
   Obs False      19198      3847
   Obs True        2903      3707
```

```r
#ConfMatrix_func(loans$Default,loans$PredProbs1,0.15)
#ConfMatrix_func(loans$Default,loans$PredProbs1,0.05)
```

Now, calculate the performance measures of the confusion matrix:

```r
predicted_value<-loans$PredDefault1
actual_value<-loans$Default

#True Negative means it was predicted negative and it is indeed negative.
TN<-sum(predicted_value==0 & actual_value==0)
#False Negative means it was predicted negative and it is actually positive.
FN<-sum(predicted_value==0 & actual_value==1)
#False Positive means it was predicted true and it is actually false.
FP<-sum(predicted_value==1 & actual_value==0)
#True Positive means it was predicted true and it is indeed true.
TP<-sum(predicted_value==1 & actual_value==1)

# Note that the logical operator "&" is used for an eval at the level of each element of th


print(paste("There are",TN," True Negatives"))
```

```
[1] "There are 19198  True Negatives"
```

```r
print(paste("There are",FN," False Negatives"))
```

```
[1] "There are 2903  False Negatives"
```

```r
print(paste("There are",FP," False Positives"))
```

```
[1] "There are 3847  False Positives"
```

```r
print(paste("There are",TP," True Positives"))
```

```
[1] "There are 3707  True Positives"
```

```r
print("As a result:")
```

```
[1] "As a result:"
```

```r
#Getting Accuracy, Sensitivity, Specificity and Precision:
#Accuracy is the number of correct predictions over the total
ACCU<-(TN+TP)/(TN+TP+FN+FP)
#True Positive Rate (TPR), also called Sensitivity or recall is the number of correct posit
TPR<-TP/(TP+FN)
#True Negative Rate (TFR), also called Specificityis the number of correct negative predict
TNR<-TN/(TN+FP)
#Precision, also called Positive Predictive Value (PPV),is the number of TP over the total
PPV<-(TP)/(TP+FP)
print(paste("Accuracy is:", round(ACCU,4)))
```

```
[1] "Accuracy is: 0.7724"
```

```r
print(paste("True Positive Rate is:", round(TPR,4)))
```

```
[1] "True Positive Rate is: 0.5608"
```

```r
print(paste("True Negative Rate is:", round(TNR,4)))
```

```
[1] "True Negative Rate is: 0.8331"
```

```r
print(paste("Precision PPV is:", round(PPV,4)))
```

```
[1] "Precision PPV is: 0.4907"
```

As you can imagine, there are packages that include functions to directly do what we did 'manually' coded above. In this case, the package is caret.

However, you need to have created the predicted values based on your selected cutoff PRIOR to calling this function.

```r
# the PredDefault1 was calculated already with a chosen Cutoff
confusionMatrix(data=as.factor(loans$PredDefault1),reference=as.factor(loans$Default))
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 19198  2903
```

```
         1  3847  3707

               Accuracy : 0.7724
                 95% CI : (0.7676, 0.7771)
    No Information Rate : 0.7771
    P-Value [Acc > NIR] : 0.9748

                  Kappa : 0.3748

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8331
            Specificity : 0.5608
         Pos Pred Value : 0.8686
         Neg Pred Value : 0.4907
             Prevalence : 0.7771
         Detection Rate : 0.6474
   Detection Prevalence : 0.7453
      Balanced Accuracy : 0.6969

       'Positive' Class : 0
```

What?? These numbers are completely different! Why? Let's look at the help file (remember this is in the Caret Package)

```
# the table is flipped, AND the 0 and 1 are reversed too!!
# the parameter "positive" might be able to help us - let's try

confusionMatrix(data=as.factor(loans$PredDefault1),reference=as.factor(loans$Default), posi
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 19198  2903
         1  3847  3707

               Accuracy : 0.7724
                 95% CI : (0.7676, 0.7771)
    No Information Rate : 0.7771
    P-Value [Acc > NIR] : 0.9748
```

```
                 Kappa : 0.3748

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.5608
            Specificity : 0.8331
         Pos Pred Value : 0.4907
         Neg Pred Value : 0.8686
             Prevalence : 0.2229
         Detection Rate : 0.1250
   Detection Prevalence : 0.2547
      Balanced Accuracy : 0.6969

       'Positive' Class : 1
```

That looks better!!

We can also just keep creating our own functions for our own purposes, in this case passing a Cutoff to the function call:

```
MyConfMatrixValues_func<-function(actual_value,predicted_prob,cutoff){

  predicted_value<-ifelse(predicted_prob>=cutoff,1,0)   # Apply the Cutoff

  ConfMatrix<-table(actual_value,predicted_value)
  rownames(ConfMatrix)<-c("Obs False", "Obs True")
  colnames(ConfMatrix)<-c("Pred False", "Pred True")

  print(paste("For a cutoff of", cutoff,":"))
  print("Actual values in the test dataset")
  print(table(actual_value))
  print("Predicted values in the test dataset")
  print(table(predicted_value))

  #predicted_value<-ifelse(predicted_prob>=cutoff,1,0)
  #True Negative means it was predicted negative and it is indeed negative.
  TN<-sum(predicted_value==0 & actual_value==0)
  #False Negative means it was predicted negative and it is actually positive.
  FN<-sum(predicted_value==0 & actual_value==1)
  #False Positive means it was predicted true and it is actually false.
  FP<-sum(predicted_value==1 & actual_value==0)
```

```
  #True Positive means it was predicted true and it is indeed true.
  TP<-sum(predicted_value==1 & actual_value==1)

  print(paste("There are",TN," True Negatives"))
  print(paste("There are",FN," False Negatives"))
  print(paste("There are",FP," False Positives"))
  print(paste("There are",TP," True Positives"))

#Getting Accuracy, Sensitivity, Specificity and Precision:
#Accuracy is the number of correct predictions over the total
  ACCU<-(TN+TP)/(TN+TP+FN+FP)
#True Positive Rate (TPR), also called Sensitivity or recall is the number of correct posit
  TPR<-TP/(TP+FN)
#True Negative Rate (TFR), also called Specificityis the number of correct negative predict
  TNR<-TN/(TN+FP)
#Precision, also called Positive Predictive Value (PPV),is the number of TP over the total
  PPV<-(TP)/(TP+FP)

  print("As a result:")
  print(paste("Accuracy is:", round(ACCU,4)))
  print(paste("True Positive (Sensitivity) Rate is:", round(TPR,4)))
  print(paste("True Negative (Specificity) Rate is:", round(TNR,4)))
  print(paste("Precision PPV is:", round(PPV,4)))

    ConfMatrixValues<-data.frame(TN,FN,FP,TP,ACCU,TPR,TNR)
  return(ConfMatrix)
}
```

Now that we defined the function, let's use it:

```
MyConfMatrixValues_func(loans$Default,loans$PredProbs1,0.25)
```

```
[1] "For a cutoff of 0.25 :"
[1] "Actual values in the test dataset"
actual_value
    0     1
23045  6610
[1] "Predicted values in the test dataset"
predicted_value
    0     1
22101  7554
```

```
[1] "There are 19198  True Negatives"
[1] "There are 2903  False Negatives"
[1] "There are 3847  False Positives"
[1] "There are 3707  True Positives"
[1] "As a result:"
[1] "Accuracy is: 0.7724"
[1] "True Positive (Sensitivity) Rate is: 0.5608"
[1] "True Negative (Specificity) Rate is: 0.8331"
[1] "Precision PPV is: 0.4907"
```

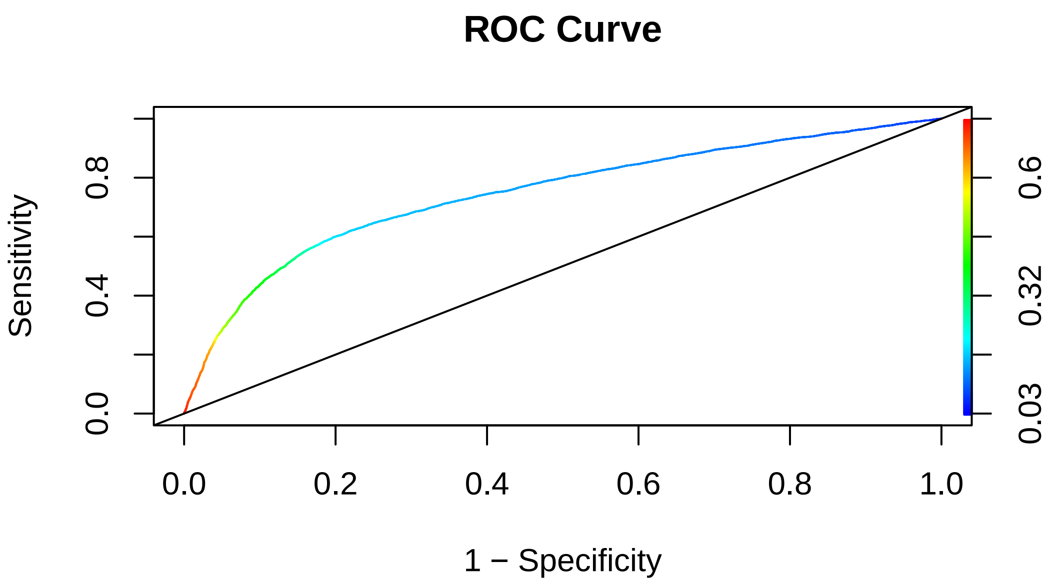
```
            predicted_value
actual_value Pred False Pred True
   Obs False      19198      3847
   Obs True        2903      3707
```

ROC (Receiver Operating Characteristic) Curve

Instead of manually trying different cutoff points, we can use the ROC Curve.
This also gives us the Area Under the Curve (AUC), which we can use to compare the model performance with other models:

```
# We'll use the function "prediction" (note: NOT predict),
# which transforms the predicted probabilities (first argument)
# and the actual 0/1 values (second argument)
# into a standardized format of class prediction, and store them into
# the object roc.pred...

roc.pred = prediction(loans$PredProbs1, loans$Default)
# ... which we can then use to actually create the ROC curve
# with the function "performance" (note: we need to store this
# so that we can then draw the curve):
perf = performance(roc.pred, "tpr", "fpr")
#If you don't get what it is doing: ?performance


plot(perf,                          # the data
     main = "ROC Curve",            # the chart's title
     xlab = "1 - Specificity",      # the name of the x-axis
     ylab = "Sensitivity",          # the name of the y-axis
     colorize=TRUE)                 # add color to curve depending on threshold prob.

# ... and add the diagonal corresponding to the Random Assignment
```

```
# benchmark model:

abline(0,1) # adds line at intercept 0, with slope 1
```




```
perf_auc = performance(roc.pred, "auc")
as.numeric(perf_auc@y.values)
```

```
[1] 0.7454106
```

That is, our first model; logreg, has an area under the curve of 0.745. How does that compare to the baseline? (AUC=0.5).

How can we increase specificity? Let's say we want a specificity of at least 90%.

```
#That puts us in the green area of about 0.4
MyConfMatrixValues_func(loans$Default,loans$PredProbs1,0.4)
```

```
[1] "For a cutoff of 0.4 :"
[1] "Actual values in the test dataset"
actual_value
    0     1
23045  6610
[1] "Predicted values in the test dataset"
predicted_value
```

```
    0     1 
24825  4830 
[1] "There are 20955  True Negatives"
[1] "There are 3870  False Negatives"
[1] "There are 2090  False Positives"
[1] "There are 2740  True Positives"
[1] "As a result:"
[1] "Accuracy is: 0.799"
[1] "True Positive (Sensitivity) Rate is: 0.4145"
[1] "True Negative (Specificity) Rate is: 0.9093"
[1] "Precision PPV is: 0.5673"
```

```
            predicted_value
actual_value Pred False Pred True
   Obs False      20955      2090
   Obs True        3870      2740
```

Comparing Models

Let's make another model with no demographic factors to avoid discrimination. Let's see the original and the new model.

```r
summary(logreg)
```

```
Call:
glm(formula = Default ~ Limit + as.factor(Gender) + as.factor(MaritalStatus) + 
    Age + Late1 + Late2 + Late3, family = "binomial", data = loans)

Coefficients:
                            Estimate Std. Error z value Pr(>|z|)    
(Intercept)               -1.5306414  0.0838129 -18.263  < 2e-16 ***
Limit                     -0.0019926  0.0001346 -14.800  < 2e-16 ***
as.factor(Gender)2        -0.1372312  0.0313593  -4.376 1.21e-05 ***
as.factor(MaritalStatus)2 -0.1661281  0.0346078  -4.800 1.58e-06 ***
Age                        0.0041690  0.0018393   2.267   0.0234 *  
Late1                      1.3578076  0.0412525  32.915  < 2e-16 ***
Late2                      0.2976810  0.0550675   5.406 6.45e-08 ***
Late3                      0.7161429  0.0472809  15.147  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 31467  on 29654  degrees of freedom
Residual deviance: 26975  on 29647  degrees of freedom
AIC: 26991

Number of Fisher Scoring iterations: 4
```

```
logreg2 = glm(Default ~ Limit + Late1 + Late2 + Late3, data=loans, family="binomial")
summary(logreg2)
```

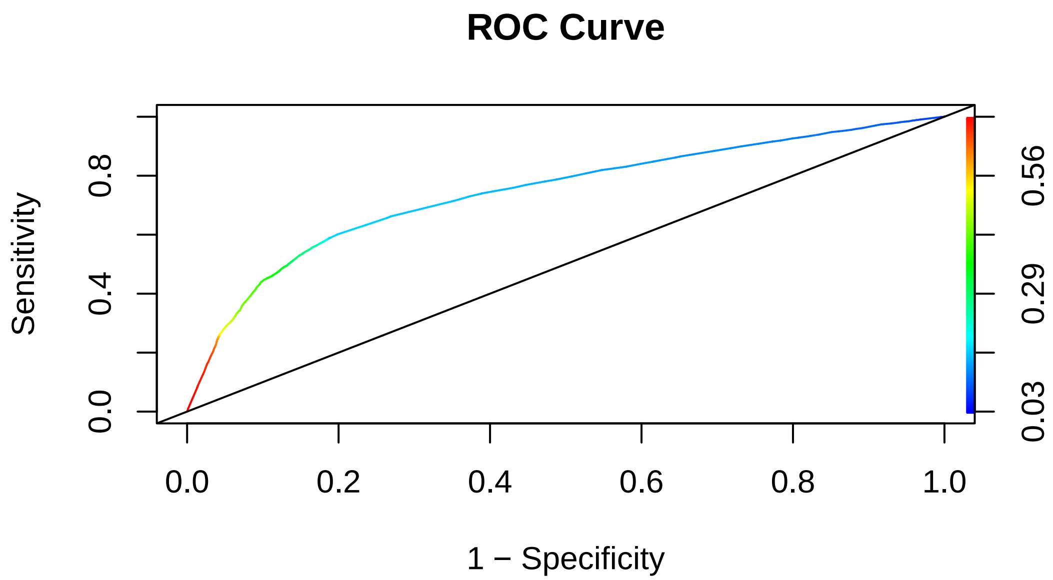
```
Call:
glm(formula = Default ~ Limit + Late1 + Late2 + Late3, family = "binomial",
    data = loans)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.572950   0.028831  -54.56  < 2e-16 ***
Limit       -0.001878   0.000133  -14.12  < 2e-16 ***
Late1        1.357784   0.041178   32.97  < 2e-16 ***
Late2        0.307697   0.054949    5.60 2.15e-08 ***
Late3        0.718916   0.047186   15.24  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 31467  on 29654  degrees of freedom
Residual deviance: 27042  on 29650  degrees of freedom
AIC: 27052

Number of Fisher Scoring iterations: 4
```

What do you see in the results? What does the AIC tell you?

Let's find the new model's ROC and AUC next:

```
#1. Make predictions:
loans$PredProbs2<-predict(logreg2, newdata=loans, type="response")
```

```r
#2 Get the ROC and AUC:
roc.pred = prediction(loans$PredProbs2, loans$Default)
# ... which we can then use to actually create the ROC curve
# with the function "performance" (note: we need to store this
# so that we can then draw the curve):
perf = performance(roc.pred, "tpr", "fpr")
#If you don't get what it is doing: ?performance


plot(perf,                              # the data
     main = "ROC Curve",                # the chart's title
     xlab = "1 - Specificity",          # the name of the x-axis
     ylab = "Sensitivity",              # the name of the y-axis
     colorize=TRUE)                     # add color to curve depending on threshold prob.

# ... and add the diagonal corresponding to the Random Assignment
# benchmark model:

abline(0,1) # adds line at intercept 0, with slope 1
```




```r
perf_auc = performance(roc.pred, "auc")
as.numeric(perf_auc@y.values)
```

```
[1] 0.7429727
```

How would you characterize the AUC difference between the models?

Which on should we use?

```
MyConfMatrixValues_func(loans$Default,loans$PredProbs2,cutoff=0.55)
```

```
[1] "For a cutoff of 0.55 :"
[1] "Actual values in the test dataset"
actual_value
    0     1
23045  6610
[1] "Predicted values in the test dataset"
predicted_value
    0     1
26956  2699
[1] "There are 22060  True Negatives"
[1] "There are 4896  False Negatives"
[1] "There are 985  False Positives"
[1] "There are 1714  True Positives"
[1] "As a result:"
[1] "Accuracy is: 0.8017"
[1] "True Positive (Sensitivity) Rate is: 0.2593"
[1] "True Negative (Specificity) Rate is: 0.9573"
[1] "Precision PPV is: 0.6351"
```

```
            predicted_value
actual_value Pred False Pred True
   Obs False      22060       985
   Obs True        4896      1714
```

Train and Test datasets

So far, we have been training and testing in the same dataset.

Usually, we would train the model on a dataset (or a portion of the data we have) and test it in another dataset (or another portion). These portions are often called "partitions"

```
#Our split should be random but we would also like to have the same "random" results
#By setting a seed we ensure everyone using the same seed gets the same "random" results.

set.seed(1020, sample.kind = "Rejection")
df<-loans
```

```r
spl = sample(nrow(df),0.8*nrow(df))
head(spl)
```

```
[1]  1360  7166  1394 13548   368 15061
```

```r
# Now lets split our dataset into train and test:

train.df = df[spl,]
test.df = df[-spl,]
dim(df)
```

```
[1] 29655    11
```

```r
dim(train.df)
```

```
[1] 23724    11
```

```r
dim(test.df)
```

```
[1] 5931   11
```

With this approach, we would train the model on the train portion of the dataset:

```r
logreg2b = glm(Default ~ Limit + Late1 + Late2 + Late3, data=train.df, family="binomial")
summary(logreg2b)
```

```
Call:
glm(formula = Default ~ Limit + Late1 + Late2 + Late3, family = "binomial",
    data = train.df)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.5720841  0.0321928 -48.833  < 2e-16 ***
Limit       -0.0018894  0.0001493 -12.652  < 2e-16 ***
Late1        1.3746490  0.0462526  29.720  < 2e-16 ***
Late2        0.2658272  0.0616894   4.309 1.64e-05 ***
Late3        0.7386307  0.0528218  13.983  < 2e-16 ***
```

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

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 25154  on 23723  degrees of freedom
Residual deviance: 21624  on 23719  degrees of freedom
AIC: 21634

Number of Fisher Scoring iterations: 4
```

Then, we can test it with a test dataset:

```
test.df$PredProbs2b<-predict(logreg2b, newdata=test.df, type="response")
MyConfMatrixValues_func(test.df$Default,test.df$PredProbs2b,cutoff=0.55)
```

```
[1] "For a cutoff of 0.55 :"
[1] "Actual values in the test dataset"
actual_value
   0    1 
4601 1330 
[1] "Predicted values in the test dataset"
predicted_value
   0    1 
5392  539 
[1] "There are 4402  True Negatives"
[1] "There are 990  False Negatives"
[1] "There are 199  False Positives"
[1] "There are 340  True Positives"
[1] "As a result:"
[1] "Accuracy is: 0.7995"
[1] "True Positive (Sensitivity) Rate is: 0.2556"
[1] "True Negative (Specificity) Rate is: 0.9567"
[1] "Precision PPV is: 0.6308"

            predicted_value
actual_value Pred False Pred True
   Obs False       4402       199
   Obs True         990       340
```

How do you find the performance to fare between Training and Testing?

Regression analysis are statistically robust, but that will no longer be the case when we move to decision trees, which are more prone to overfitting.