Exercise 9 - Classification Models

O’Malley, Conie

2025-02-26

Table of Contents

## Preparation

Download the **Ex9\_Classification\_YourLastName.Rmd** R Markdown file to your working directory, rename it to include your last name and follow the instructions below. When you finish, knit it into an HTML file and upload it (the zipped or PDF version of your knitted file) onto Blackboard.

## 1. Data Work

**1.1 Read Data**

Download the **myopia.csv** file from Blackboard to your working directiory and read it using read.table() with the parameters header=T, row.names=1, sep=",". Store the dataset in an object named **myopia**. For sanity check, display the first 10 rows and 6 columns **myopia** data frame.

Dataset documentation at: <https://rdrr.io/cran/aplore3/man/myopia.html> Please note that **myopic** is coded as 1 (Yes), 0 (No) (not 1 and 2)

myopia <- utils::read.table("/Users/coniecakes/Library/CloudStorage/OneDrive-Personal/001. Documents - Main/023. Programming Tools/R Studio/PredictiveAnalytics/R\_Exercises/data/myopia.csv", header=T, row.names=1, sep=",")   
myopia[1:10, 1:6]

## study.year myopic age female spheq al  
## 1 1992 1 6 1 -0.052 21.89  
## 2 1995 0 6 1 0.608 22.38  
## 3 1991 0 6 1 1.179 22.49  
## 4 1990 1 6 1 0.525 22.20  
## 5 1995 0 5 0 0.697 23.29  
## 6 1995 0 6 0 1.744 22.14  
## 7 1993 0 6 1 0.683 22.33  
## 8 1991 0 6 1 1.272 22.39  
## 9 1991 0 7 0 1.396 22.62  
## 10 1991 0 6 1 0.972 22.74

**1.2 Train Vector**

Enter RNGkind(sample.kind="default"), set the **seed** at 1 and create a random train vector with **80%** of the row numbers in **myopia**. Display the first 100 elements of **train** to compare with the result below.

\*\*Note:\* your numbers should match mine, but they may vary slightly do to sampling rounding, which is OK. Notice that my display of the vector has 17 values in each row. Sometimes students get fewer or more values per row and wonder if their index vector is wrong. Again, you should compare the nth. value in my index vector to the nth. value in yours. And again, small differences don’t matter.

RNGkind(sample.kind="default")  
set.seed(1)   
train=sample(nrow(myopia), 0.8\*nrow(myopia))   
train[1:100]

## [1] 129 509 471 299 270 187 307 597 277 494 330 591 37 105 485 382 601 326  
## [19] 608 554 422 111 404 532 506 556 343 582 121 40 537 375 248 198 378 39  
## [37] 435 390 280 526 45 402 22 193 371 499 104 602 492 465 525 176 345 110  
## [55] 84 29 141 252 304 545 557 287 145 329 487 498 490 103 316 51 290 618  
## [73] 282 143 442 285 48 501 511 295 536 214 339 346 43 1 563 233 293 369  
## [91] 451 86 483 327 355 49 361 550 242 440

**1.3 Train and Test Sub-Samples**

Create the corresponding **myopia.train**  and **myopia.test** subsamples, using the [train,] and [-train,] indices respectively. Save these subsamples with the names **myopia.train** and **myopia.test** respectively. For sanity check, display their respectiv nrow() counts.

myopia.train <- myopia[train,]   
myopia.test <- myopia[-train,]   
nrow(myopia.train)

## [1] 494

nrow(myopia.test)

## [1] 124

## 2. Binomial Logistic Model

**2.1 Fit the Logistic Model**

Fit a logistic model to predict whether a kid is **myopic**, using age + female + sports.hrs + read.hrs + mommy + dadmy as predictors. Fit the model on the **myopia.train** subset. Store the results in an object named **glm.fit.train**. Display the **summary()** results. **Tip:** use family="binomial"(link="logit"). Then display the summary() results.

glm.fit.train <- stats::glm(myopic ~ age + female + sports.hrs + read.hrs + mommy + dadmy,   
 family = "binomial"(link="logit"), data = myopia.train)  
  
summary(glm.fit.train)

##   
## Call:  
## stats::glm(formula = myopic ~ age + female + sports.hrs + read.hrs +   
## mommy + dadmy, family = binomial(link = "logit"), data = myopia.train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -8.53150 2.53689 -3.363 0.000771 \*\*\*  
## age -0.03972 0.34185 -0.116 0.907499   
## female -0.02108 0.50713 -0.042 0.966851   
## sports.hrs -0.13714 0.03664 -3.743 0.000182 \*\*\*  
## read.hrs 0.75784 0.10281 7.371 1.69e-13 \*\*\*  
## mommy 2.77370 0.58246 4.762 1.92e-06 \*\*\*  
## dadmy 2.79940 0.59202 4.729 2.26e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 392.18 on 493 degrees of freedom  
## Residual deviance: 110.14 on 487 degrees of freedom  
## AIC: 124.14  
##   
## Number of Fisher Scoring iterations: 8

2.2 For interpretation purposes, display the log-odds alongside the odds.

log.odds <- coef(glm.fit.train) # To get just the coefficients  
odds <- exp(log.odds) # To convert log-odds to odds  
  
options(scipen=4) # Limit use of scientific notation  
print(cbind("Log-Odds"=log.odds, "Odds"=odds), digits=2) # All together

## Log-Odds Odds  
## (Intercept) -8.532 0.0002  
## age -0.040 0.9611  
## female -0.021 0.9791  
## sports.hrs -0.137 0.8718  
## read.hrs 0.758 2.1337  
## mommy 2.774 16.0178  
## dadmy 2.799 16.4348

2.3 Provide a brief interpretation of the effect of sports.hrs, read.hrs and having myopic parents.

#### Analysis

* sports.hrs - , , which means that for every additional hour spent on sports, the odds of having myopia decrease by 13.3%.
* read.hrs - , , which means that for every additional hour spent reading, the odds of having myopia increase by 113.37%.
* myopic parents
  + mommy - , , which means that for every additional hour spent reading, the odds of having myopia increase by 1501.78%.
  + dadmy - , , which means that for every additional hour spent reading, the odds of having myopia increase by 1534.48%.

## 3. Classification Predictions

**3.1 Test Predictions**

Use the **glm.fit.train** model to precict probabilities using the **myopia.test** data. Use the predict() function, with the **glm.fit.train** model, and the **myopia.test** dataset, with a type="response", which will yield probabilities. Store the results in an object named **probs.test**. Display the first 10 predicted probability values (of being a 1) in **probs.test**.

Then use the ifelse() function to convert these probabilities into actual predictions, yielding 1 for probs.test>0.5, 0 otherwise. Store the results in an object named **pred.test**. Use the cbind() to display the first 20 records of **probs.test** along with their corresponding vallue in **pred.test**. Note that all probabilities > 0.5 have a 1 and the others a 0.

probs.test <- stats::predict(glm.fit.train, myopia.test, type="response")   
pred.test <- ifelse(probs.test>0.5, 1, 0)   
cbind(probs.test, pred.test)[1:20,]

## probs.test pred.test  
## 9 0.00008625528 0  
## 12 0.05274981918 0  
## 13 0.00023604017 0  
## 17 0.00387138670 0  
## 23 0.00294073017 0  
## 24 0.00446955264 0  
## 59 0.00161925364 0  
## 61 0.02987242104 0  
## 63 0.00017942864 0  
## 66 0.76077250857 1  
## 67 0.00003941853 0  
## 68 0.98026168877 1  
## 70 0.00517173683 0  
## 74 0.00011807047 0  
## 76 0.00286633518 0  
## 80 0.00016694291 0  
## 82 0.00008787851 0  
## 88 0.00612406028 0  
## 94 0.00100617419 0  
## 95 0.00016094054 0

## 4. Confusion Matrix

**4.1 Build the Confusion Matrix**

Build a confusion matrix using the table() function. Use these two vectors in the table() function: "Predicted"=pred.test (which contains your model predictions) and "Actual"=myopia.test$myopic (which contains the actual data in the test subset). Store your confusion matrix results in an object named **conf.mat** and display it.

conf.mat <- table("Predicted"=pred.test, "Actual"=myopia.test$myopic)   
conf.mat

## Actual  
## Predicted 0 1  
## 0 109 3  
## 1 1 11

\*\*4.2 Accuracy Statistics

Compute all the classification statistics displayed below. Given that we named to confusion matrix **conf.mat** you should be able to just copy and paste the commands in the scripts provided for class:

# Done for you, but you need to continue below  
  
# Counting cell values from the confusion matrix  
  
TruN <- conf.mat[1,1] # True negatives, row 1 col 1  
TruP <- conf.mat[2,2] # True positives, row 2 col 2  
FalP <- conf.mat[2,1] # False positives, etc.  
FalN <- conf.mat[1,2] # False negatives  
TotN <- conf.mat[1,1] + conf.mat[2,1] # Total negatives  
TotP <- conf.mat[1,2] + conf.mat[2,2] # Total positives  
Tot <- TotN+TotP # Matrix total  
  
# Now continue on your own:  
model\_accuracy <- (TruN+TruP)/Tot  
model\_error <- (FalN+FalP)/Tot   
model\_sensitivity <- TruP/TotP   
model\_specificity <- TruN/TotN   
fp\_rate <- 1-model\_specificity

**4.3 Display Accuracy Statistics**

Use the c() function to create a vector with the results above. Call this vector **logit.rates.50**. Then use the names() function to provide the corresponding labels, as shown below. Also, use the print() function with 2 digits to display the results.

logit.rates.50 <- c(model\_accuracy, model\_error, model\_sensitivity, model\_specificity, fp\_rate)   
names(logit.rates.50) <- c("Accuracy", "Error", "Sensitivity", "Specificity", "False Positives")   
print(logit.rates.50, digits=2)

## Accuracy Error Sensitivity Specificity False Positives   
## 0.9677 0.0323 0.7857 0.9909 0.0091

4.4 Provide a brief commentary on these results.

#### Analysis

* Accuracy - 96.77%, indicating that a high proportion of total predictions made by the model are correct. This suggests the model is generally reliable in distinguishing between classes.
* Error Rate = 3.23%, is relatively low, indicating that only a small fraction of predictions deviate from the true labels. This reinforces the model’s overall effectiveness.
* Sensitivity = 78.57%, implies that the model is moderately good at identifying positive cases, but there is room for improvement to better capture all true positive instances.
* Specificity = 99.09%, demonstrates that the model is highly effective in accurately identifying negative cases, minimizing the number of false alarms or false positives.
* False Positives = 00.91%, the model makes very few incorrect positive predictions, which is beneficial for maintaining the credibility of positive class predictions.

4.5 Repeat 3.1, 3.2 and 3.3 but this time decrease the classification probability threshold from 0.5 to **0.3**. Note how the confusion matrix and classification statistics change.

# Done for you  
  
# Classification threshold: 30%  
  
pred.test <- ifelse(probs.test>0.3, 1,0)  
  
conf.mat <- table("Predicted"=pred.test, "Actual"=myopia.test$myopic)   
conf.mat

## Actual  
## Predicted 0 1  
## 0 106 1  
## 1 4 13

cat("\n") # Concatenate function "\n" prints a new blank line

TruN <- conf.mat[1,1] # True negatives, row 1 col 1  
TruP <- conf.mat[2,2] # True positives, row 2 col 2  
FalP <- conf.mat[2,1] # False positives, etc.  
FalN <- conf.mat[1,2] # False negatives  
TotN <- conf.mat[1,1] + conf.mat[2,1] # Total negatives  
TotP <- conf.mat[1,2] + conf.mat[2,2] # Total positives  
Tot <- TotN+TotP # Matrix total  
  
Accuracy <- (TruN+TruP)/Tot  
Error <- (FalN+FalP)/Tot  
Sensitivity <- TruP/TotP # Proportion of correct positives  
Specificity <- TruN/TotN # Proportion of correct negatives  
FalPos <- 1 - Specificity  
  
logit.rates.30=c(Accuracy, Error, Sensitivity, Specificity, FalPos)  
names(logit.rates.30)=c("Accuracy", "Error", "Sensitivity", "Specificity", "False Positives")  
print(logit.rates.30, digits=2) # Prints fewer digits

## Accuracy Error Sensitivity Specificity False Positives   
## 0.960 0.040 0.929 0.964 0.036

4.6 Then change increase the threshold to **0.8** and note the changes again. Think but don’t answer, what happened to the various statistics when you made these changes?

# Done for you  
  
# Classification threshold: 70%  
  
pred.test <- ifelse(probs.test>0.8, 1,0)  
conf.mat <- table("Predicted"=pred.test, "Actual"=myopia.test$myopic)   
conf.mat

## Actual  
## Predicted 0 1  
## 0 110 8  
## 1 0 6

cat("\n") # Concatenate function "\n" prints a new blank line

TruN <- conf.mat[1,1] # True negatives, row 1 col 1  
TruP <- conf.mat[2,2] # True positives, row 2 col 2  
FalP <- conf.mat[2,1] # False positives, etc.  
FalN <- conf.mat[1,2] # False negatives  
TotN <- conf.mat[1,1] + conf.mat[2,1] # Total negatives  
TotP <- conf.mat[1,2] + conf.mat[2,2] # Total positives  
Tot <- TotN+TotP # Matrix total  
  
Accuracy <- (TruN+TruP)/Tot  
Error <- (FalN+FalP)/Tot  
Sensitivity <- TruP/TotP # Proportion of correct positives  
Specificity <- TruN/TotN # Proportion of correct negatives  
FalPos <- 1 - Specificity  
  
logit.rates.80 <- c(Accuracy, Error, Sensitivity, Specificity, FalPos)  
names(logit.rates.80) <- c("Accuracy", "Error", "Sensitivity", "Specificity", "False Positives")  
  
print(logit.rates.80, digits=2) # Prints fewer digits

## Accuracy Error Sensitivity Specificity False Positives   
## 0.935 0.065 0.429 1.000 0.000

4.7 Now, use the rbind() function to print the results for all three sets of statistics above.

cat("\n") # Concatenate function "\n" prints a new blank line

print(rbind(logit.rates.30, logit.rates.50, logit.rates.80), digits=2)

## Accuracy Error Sensitivity Specificity False Positives  
## logit.rates.30 0.96 0.040 0.93 0.96 0.0364  
## logit.rates.50 0.97 0.032 0.79 0.99 0.0091  
## logit.rates.80 0.94 0.065 0.43 1.00 0.0000

4.8 Provide a brief discussion of your observations from the 3 sets of results above

#### Analysis

* **logit.rates.30:** At the 30% threshold level, the model demonstrates high accuracy and specificity, indicating it makes very few false positive predictions. However, with a sensitivity of 0.93, it slightly compromises on correctly identifying positive cases.
* **logit.rates.50:** At the 50% threshold, both accuracy and specificity are further enhanced, with specificity nearly perfect at 0.99, suggesting minimal false positives. However, sensitivity drops significantly to 0.79, indicating the model misses more true positives compared to the 30% threshold.
* **logit.rates.80:** With the threshold increased to 80%, the model achieves perfect specificity, effectively eliminating false positives, but sensitivity plunges to 0.43, highlighting a substantial decrease in the model’s capability to detect true positives. This threshold level seems heavily skewed towards avoiding false positives at the cost of missing a majority of positive cases.

## 5. ROC Curve and Area Under the Curve (AUC)

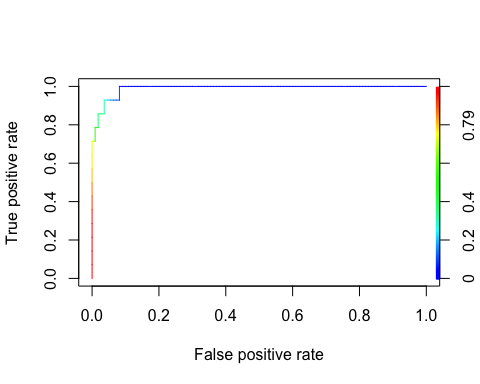
To better examine how the model performs as we change the classification threshold, let’s plot the **ROC** curve and compute the corresponding area under the curve **AUC**.

\*\*5.1 ROC Curve

Load the **{ROCR}** library, which contains the necessary functions for ROC computation. You already predicted high mileage classifications with the test subsample above and stored it in **pred.probs**. But this time we need to use the **ROCR** prediction() function to create an ROC prediction object. Use probs.test (i.e., predicted classifications in the test subsample) and myopia.test$myopic (actual classifications in the test subsample) as arguments in the prediction() function and store the results in an object named **pred**.

Then feed this **pred** object into the perf() function, but also include the "tpr" and "fpr" arguments to obtain true positive and false positive rates, respectively (which is what goes on the ROC Curve). Then plot the **perf** object using colorize=T to get a colorful plot.

library(ROCR)   
pred <- prediction(probs.test, myopia.test$myopic)   
perf <- performance(pred,"tpr","fpr")   
plot(perf, colorize=T)



**5.2 AUC**

Feed the **pred** object again into the perf() function again, but this time use "auc" as the argument, to get the area under the ROC Curve, AUC. Then display the name and value of the **AUC** with c(auc@y.name[[1]], auc@y.values[[1]])

auc <- performance(pred,"auc")   
c(auc@y.name[[1]], auc@y.values[[1]])

## [1] "Area under the ROC curve" "0.98961038961039"