Appendix R9: Classification Models

J. Alberto Espinosa

1/31/2022

Table of Contents

1. Introduction

Overview

Classification models predict outcomes that are either binary or categorical. There are several types of classification models, including Logistic regression, classification trees, linear and quadratic discriminant analysis (LDA and QDA), and classification neural networks, among others. We will be using the Stock Market **{ISLR}Smarket** data set frequently in this section. Before you continue, review the variables in this data set by typing **?Smarket** at the console.

```
library(ISLR) # Contains the Smarket data set
dim(Smarket) # Dimensions of the Smarket data set

## [1] 1250    9
```

```
# 1250 observations and 9 variables
```

This data set does not yield very interesting results in Logistic models but the results illustrate a few important points. It also illustrates the use of lag transformations in logistic models. Let's explore the data

```
names(Smarket)

## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"

head(Smarket)

##   Year   Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001  0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001  0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001  1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
## 4 2001 -0.623  1.032  0.959  0.381 -0.192 1.2760  0.614        Up
## 5 2001  0.614 -0.623  1.032  0.959  0.381 1.2057  0.213        Up
## 6 2001  0.213  0.614 -0.623  1.032  0.959 1.3491  1.392        Up
```

One important thing to note is that the variable "Direction" is categorical, not binary. So, we have two choices, create a binary variable called "Up" = 1 when Direction is up and 0 otherwise. But in the example below we don't make conversions until later and we let R take care of this for us. Let's now explore the data:

```
summary(Smarket)

##       Year           Lag1                Lag2                Lag3
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500   1st Qu.:-0.640000
##  Median :2003   Median : 0.039000   Median : 0.039000   Median : 0.038500
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919   Mean   : 0.001716
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.733000
##       Lag4                Lag5              Volume           Today
##  Min.   :-4.922000   Min.   :-4.92200   Min.   :0.3561   Min.   :-4.922000
##  1st Qu.:-0.640000   1st Qu.:-0.64000   1st Qu.:1.2574   1st Qu.:-0.639500
##  Median : 0.038500   Median : 0.03850   Median :1.4229   Median : 0.038500
##  Mean   : 0.001636   Mean   : 0.00561   Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.: 0.596750   3rd Qu.: 0.59700   3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   : 5.733000   Max.   : 5.73300   Max.   :3.1525   Max.   : 5.733000
##  Direction
##  Down:602
##  Up  :648
##
##
##
##
```

The Generalized Linear Model Function glm()

As discussed in the Basic Models chapter, Ordinary Least Squares (OLS) regressions require the residuals to be normally distributed and the outcome variable to be continuous. When these assumptions don't hold, the Generalized Linear Model (GLM) can be used instead. OLS models are estimated with the **lm()** function in R, whereas GLM models are estimated with the **glm()** function. The syntax for the **glm()** function is the same as for the **lm()**, except that we need to inform the function about two additional things:

- The **family distribution** of the residuals, specified with the `family =` parameter
- Any required **transformations** of the outcome variable to overcome the lack of continuity, specified with the `link =` function inside the family parameter.

A linear model with no transformation of the dependent variable estimated with **glm()** and a **gaussian** distribution yields the exact same results as **OLS** with the **lm()** function. But **glm()** supports other distributions and transformations of the dependent variable besides the normal distribution.

Different regression methods require specific model `family =` and `link =` parameters in the **glm()** function:

- Binomial Logistic regression – `family = binomial(link = "logit")`
- OLS regression – `family = gaussian(link = "identity")`
- Binomial Probit regression – `family = binomial(link = "probit")` - probit stands for **Probability Unit** and it works just like logit, except that it uses a different transformation function for Y.
- Count data models – `family = poisson(link = "log")`

2. Binomial Logistic Regression

In the next example, I will use the Heart.csv I had already downloaded from the ISLR website, but you could read the file directly from the web as follows:

```
heart <- read.table(
"https://web.stanford.edu/~hastie/ElemStatLearn/datasets/SAheart.data", sep =
",", head = T)
```

The data set documentation is available at:

https://web.stanford.edu/~hastie/ElemStatLearn/datasets/SAheart.info.txt

```
heart <- read.table("Heart.csv",
                    sep = ",",
                    head = T,
                    stringsAsFactors = T)

head(heart)
```

```
##   sbp tobacco  ldl adiposity famhist typea obesity alcohol age chd
## 1 160   12.00 5.73     23.11 Present    49   25.30   97.20  52   1
## 2 144    0.01 4.41     28.61  Absent    55   28.87    2.06  63   1
## 3 118    0.08 3.48     32.28 Present    52   29.14    3.81  46   0
## 4 170    7.50 6.41     38.03 Present    51   31.99   24.26  58   1
## 5 134   13.60 3.50     27.78 Present    60   25.99   57.34  49   1
## 6 132    6.20 6.47     36.21 Present    62   30.77   14.14  45   0
```

Logistic model predicting coronary heart disease:

```
heart.fit <- glm(chd ~ .,
                 family = binomial(link = "logit"),
                 data = heart)

summary(heart.fit)
```

```
##
## Call:
## glm(formula = chd ~ ., family = binomial(link = "logit"), data = heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7781  -0.8213  -0.4387   0.8889   2.5435
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -6.1507209  1.3082600  -4.701 2.58e-06 ***
## sbp             0.0065040  0.0057304   1.135 0.256374
## tobacco         0.0793764  0.0266028   2.984 0.002847 **
## ldl             0.1739239  0.0596617   2.915 0.003555 **
## adiposity       0.0185866  0.0292894   0.635 0.525700
## famhistPresent  0.9253704  0.2278940   4.061 4.90e-05 ***
## typea           0.0395950  0.0123202   3.214 0.001310 **
## obesity        -0.0629099  0.0442477  -1.422 0.155095
## alcohol         0.0001217  0.0044832   0.027 0.978350
## age             0.0452253  0.0121298   3.728 0.000193 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 472.14  on 452  degrees of freedom
## AIC: 492.14
##
## Number of Fisher Scoring iterations: 5
```

Looks like tobacco, ldl, family history, type a and age are the strongest predictors of coronary heart desease (chd). Interestingly Once you control for these factors obesity and alcohol are not significant predictors. Take a closer look:

Logistic Regression Fit Statistics

```
-2 * logLik(heart.fit) # 2LL

## 'log Lik.' 472.14 (df=10)

deviance(heart.fit) # Should yield the same value

## [1] 472.14
```

There are various measures of deviance ajusted for model size, such as the Akaike Information Criterion (**AIC**), which is equal to `2LL + 2 * Number of variables`

```
AIC(heart.fit)

## [1] 492.14
```

Let's display them all together

```
c("2LL" = -2 * logLik(heart.fit),
  "Deviance" = deviance(heart.fit),
  "AIC" = AIC(heart.fit))

##      2LL Deviance      AIC
##   472.14   472.14   492.14
```

From Log-Odds to Odds

```
log.odds <- coef(heart.fit) # Get the coefficients
odds <- exp(log.odds) # Convert log-odds to odds
```

We can also convert the odds to probabilities, but the probability effect of a coefficient is relative to the value of the predictors, so it changes as the predictors change, so they are not very useful for interpretations, other than saying that any coefficient with a probability greater than 0.5 increases the likelihood of a positive outcome, whereas less than 0.5 decreases the likelihood. Nevertheless, probabilities are very useful when predicting actual outcomes.

```
prob = odds / (1 + odds) # Convert odds to probabilities

# Display all 3 together

round(cbind("Log-Odds" = log.odds,
            "Odds" = odds,
            "Probabilities" = prob),
      digits = 3)

##                Log-Odds  Odds Probabilities
## (Intercept)      -6.151 0.002         0.002
## sbp               0.007 1.007         0.502
## tobacco           0.079 1.083         0.520
## ldl               0.174 1.190         0.543
## adiposity         0.019 1.019         0.505
## famhistPresent    0.925 2.523         0.716
## typea             0.040 1.040         0.510
## obesity          -0.063 0.939         0.484
## alcohol           0.000 1.000         0.500
## age               0.045 1.046         0.511
```

95% Confidence intervals

With log-odds:

```
round(confint(heart.fit), digits = 3) # In log-odds
```

```
##                 2.5 % 97.5 %
## (Intercept)    -8.776 -3.636
## sbp            -0.005  0.018
## tobacco         0.029  0.133
## ldl             0.059  0.294
## adiposity      -0.038  0.077
## famhistPresent  0.481  1.376
## typea           0.016  0.064
## obesity        -0.152  0.022
## alcohol        -0.009  0.009
## age             0.022  0.069
```

With odds:

```
round(exp(confint(heart.fit)), digits = 3)  # In odds
```

```
##                2.5 % 97.5 %
## (Intercept)    0.000  0.026
## sbp            0.995  1.018
## tobacco        1.029  1.142
## ldl            1.061  1.342
## adiposity      0.962  1.080
## famhistPresent 1.618  3.958
## typea          1.016  1.066
## obesity        0.859  1.022
## alcohol        0.991  1.009
## age            1.022  1.072
```

3. The Confusion Matrix

Cross-Validation Train and Test Sub-Samples

Let's do some **Cross-Validation** testing with the **Confusion Matrix**.

Let's start by defining the train and test sub-samples using the **heart** data frame. First, let's set the default random number generator and random seed (to get repeatable results)

```
RNGkind(sample.kind="default")
set.seed(1)
```

Then, let's create an index vector for the train sub-sample

```
train <- sample(1:nrow(heart), 0.7 * nrow(heart))
```

We can now split the data into train and test sub-samples

```
heart.train <- heart[train,] # Train sub-sample
nrow(heart.train) # Count the train observations

## [1] 323

heart.test <- heart[-train,] # Test sub-sample
nrow(heart.test) # Count the test observations

## [1] 139
```

We can now train a Logistic regression model usint the train sub-sample

```
heart.fit.train <- glm(chd ~ .,
                       family = binomial(link = "logit"),
                       data = heart.train)
```

We can now use the trained model to make predictions with the test data and evaluate the cross-validation accuracy of the model and build a confusion matrix with the results.

Confusion with cross-Validated Test Results

We use `type="response"` to get **probability of Y = 1**. Notice that we use the trained model, **heart.fit.train** to make the predictions, but we make the predictions using the test subsample **heart.test**.

```
heart.probs.test <- predict(heart.fit.train,
                            heart.test,
                            type = "response")

heart.probs.test[1:10] # Take a look at the first 10 predictions

##          5          6          7          8          9         10 
11
## 0.74727697 0.64619084 0.19114356 0.62648581 0.13616443 0.58022506 
0.59418432
##         17         18         21 
## 0.83491902 0.89829067 0.07528622
```

To build the confusion matrix, we need to convert the predicted probabilities computed above into actual classifications. In order to do this we need to set the classification threshold lambda. For this example, I used **lambda = 0.5**, so that any observation with a predicted probability of being a positive (i.e., 1) is greater than 0.5 or 50%. We can alter this tuning parameter later.

```
lambda <- 0.5
```

Notice that I stored the threshold value in a variable named **lambda**. This way I can use the variable rather than the actual value 0.5 in the formulas below. This way, if I wish to change the threshold to another value, I just change it above and all the formulas that follow will work. You can try several values of lambda on your own.

Let's create a vector to store a binary value 1 if the probability of the outcome being 1 exceeds this threshold and 0 otherwise. Let's then display the probabilities of the outcome variable **chd** being 1 side by side with the actual classifications. All probabilities > 0.5 should have a 1, and a 0 otherwise.

```
heart.pred.test <- ifelse(heart.probs.test > lambda, 1, 0)

cbind("Prob chd = 1" = round(heart.probs.test, digits = 3),
      "Predicted Classification" = heart.pred.test)[1:10,]

##    Prob chd = 1 Predicted Classification
## 5         0.747                        1
## 6         0.646                        1
## 7         0.191                        0
## 8         0.626                        1
## 9         0.136                        0
## 10        0.580                        1
## 11        0.594                        1
## 17        0.835                        1
## 18        0.898                        1
## 21        0.075                        0
```

Cross-Tabulation for the Confusion Matrix

Technical Note: The construction of the confusion matrix that follows is a bit involved, but notice that once you do the cross-tabulation with the results of any binomial classification model, if you name your cross-table **conf.mat**, the rest of the script lines will work without modification. Just copy/paste.

```
conf.mat <- table("Predicted" = heart.pred.test,
                  "Actual" = heart.test$chd)

conf.mat

##          Actual
## Predicted  0  1
##         0 74 26
##         1 16 23
```

Confusion Matrix Metrics

```
TruN <- conf.mat[1,1] # True negatives
TruP <- conf.mat[2,2] # True positives
FalN <- conf.mat[1,2] # False negatives
```

```
FalP <- conf.mat[2,1] # False positives
TotN <- TruN + FalP # Total actual negatives
TotP <- TruP + FalN # Total actual positives
TotNpr <- TruN + FalN # Total negative predictions
TotPpr <- TruP + FalP # Total positive predictions
Tot <- TotN + TotP # Total

# Do a quick check of the computations
cbind(TruN, TruP, FalN, FalP, TotN, TotP, TotNpr, TotPpr, Tot)

##      TruN TruP FalN FalP TotN TotP TotNpr TotPpr Tot
## [1,]   74   23   26   16   90   49    100     39 139
```

Let’s add totals and labels to the confusion matrix

```
conf.mat.totals <- cbind(conf.mat, c(TotNpr, TotPpr))
conf.mat.totals <- rbind(conf.mat.totals, c(TotN, TotP, Tot))

colnames(conf.mat.totals) <-
  rownames(conf.mat.totals) <-
     c("No","Yes", "Total")

conf.mat.totals

##       No Yes Total
## No    74  26   100
## Yes   16  23    39
## Total 90  49   139
```

Confusion Matrix Accuracy and Error Rates

```
# Aggregate rates
Accuracy.Rate <- (TruN + TruP) / Tot
Error.Rate <- (FalN + FalP) / Tot

# Sensitivity -- rate of correct positives
Sensitivity <- TruP / TotP

# Specificity -- rate of correct negatives
Specificity <- TruN / TotN

# False Positive Rate = 1 - specificity (useful for ROC curves)
FalseP.Rate <- 1 - Specificity

# Let's combine them in a vector

logit.rates.50 <- c(Accuracy.Rate, Error.Rate, Sensitivity,
                    Specificity, FalseP.Rate)

names(logit.rates.50) <- c("Accuracy", "Error", "Sensitivity",
                           "Specificity", "False Pos")

print(logit.rates.50, digits=2)

##    Accuracy       Error Sensitivity Specificity   False Pos
##        0.70        0.30        0.47        0.82        0.18
```

Change the Threshold

To illustrate how simple it is to change the threshold, I have copied all the instructions above in to the script lines that follow, except that I set the lambda threshold value to **0.6**. Two important technical issues about this: (1) if you use a different model and data set, all you need to do is construct the **conf.mat** confusion matrix and the rest of the script commands will work; (2) if you have experience programming, you can easily embed these script lines inside a loop function that can compute confusion matrices for several lambda values. You should try it.

```
lambda <- 0.6
heart.pred.test <- ifelse(heart.probs.test > lambda, 1, 0)

cbind("Prob chd = 1" = round(heart.probs.test, digits = 3),
      "Predicted Classification" = heart.pred.test)[1:10,]

##    Prob chd = 1 Predicted Classification
## 5         0.747                        1
## 6         0.646                        1
## 7         0.191                        0
## 8         0.626                        1
## 9         0.136                        0
## 10        0.580                        0
## 11        0.594                        0
## 17        0.835                        1
## 18        0.898                        1
## 21        0.075                        0

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

TruN   <- conf.mat[1,1];  TruP <- conf.mat[2,2]
FalN   <- conf.mat[1,2];  FalP <- conf.mat[2,1]
TotN   <- TruN + FalP;    TotP <- TruP + FalN
TotNpr <- TruN + FalN;  TotPpr <- TruP + FalP
Tot    <- TotN + TotP

conf.mat.totals <- cbind(conf.mat, c(TotNpr, TotPpr))
conf.mat.totals <- rbind(conf.mat.totals, c(TotN, TotP, Tot))

colnames(conf.mat.totals) <-
  rownames(conf.mat.totals) <-
     c("No","Yes", "Total")
```

```
conf.mat.totals

##       No Yes Total
## No    78  33   111
## Yes   12  16    28
## Total 90  49   139

Accuracy.Rate <- (TruN + TruP) / Tot
Error.Rate <- (FalN + FalP) / Tot
Sensitivity <- TruP / TotP
Specificity <- TruN / TotN
FalseP.Rate <- 1 - Specificity

logit.rates.60 <- c(Accuracy.Rate, Error.Rate, Sensitivity,
                    Specificity, FalseP.Rate)

names(logit.rates.60) <- c("Accuracy", "Error", "Sensitivity",
                           "Specificity", "False Pos")

# Compare the rates for both thresholds and see the sharp differences

logit.rates.compare <- rbind(logit.rates.50, logit.rates.60)

print(logit.rates.compare, digits = 2)

##                Accuracy Error Sensitivity Specificity False Pos
## logit.rates.50     0.70  0.30        0.47        0.82      0.18
## logit.rates.60     0.68  0.32        0.33        0.87      0.13
```

4. The Receiver Operating Characteristics (ROC) Curve

To render ROC curves we use the **ROCR** library:

```
library(ROCR)
```

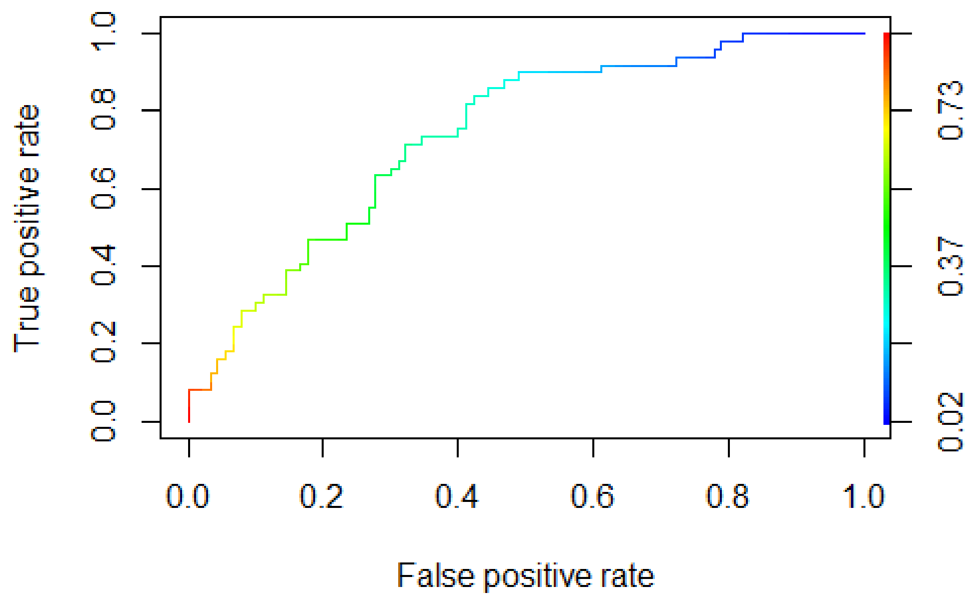
The first step with ROCR is to create a **prediction()** object. It is important to note that this function is contained in the **{ROCR}** library and it is NOT the same as the **predict()** function from the **{stat}** package. The predict() function is used to make predictions, whereas the prediction() function is used to render the ROC curve. The prediction() function uses two vectors:

- The predicted probabilities for the test sub-sample, as computed above, and
- The corresponding actual values of the outcome variable in the test sub-sample

```
pred <- prediction(heart.probs.test, heart.test$chd)
```

The next step is to use the **{ROCR}performance()** function to create the ROC object. Use the parameter "`tpr`" for **True Positive Rate** in the vertical axis, and "`fpr`" for **False Positive Rate** in the horizontal axis. Other possible values are: "`acc`" for accuracy; "`err`" for error rate; "sens" for sensitivity; "spec" fpr specificity; and "auc"for area under the curve.

```
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = T) # Plot the ROC curve with colors
```



Computing the AUC

```
auc <- performance(pred, "auc")
```

The performance() object above stores the **name** of the variable in **@y.name [[1]]** and the actual **AUC** in **@y.values [[1]]**. Note: the use of double brackets [[]] instead of single brackets [] and @ instead of $ to access values is because the performance() object is a “list” not a data frame. Lists use [[]] for indexing values and @ for accessing elements in the list. Also, note that I round the AUC value to keep it short.

```
auc.name <- auc@y.name[[1]] # Retrieve the AUC name from the AUC list
auc.value <- round(auc@y.values[[1]], digits = 3) # Retrieve the AUC value

paste(auc.name, "is", auc.value) # Display them together

## [1] "Area under the ROC curve is 0.739"
```

5. Mutinomial Logistic Regression

A multinomial Logistic model is identical to a binomial Logistic model, except that the categorical outcome variable has more than 2 possible values. For example, suppose that you are trying to predict the likelihood that real estate clients will buy rural, suburban or urban homes. In this case, the dependent variable is categorical and has has 3 possible values. This model can be easily fitted using 2 binomial Logistic models.

First, you would need to select a baseline or reference value (e.g., rural) and then create dummy variables for the two other categories (i.e., suburban and urban). You then would need to build a binomial Logistic model with `suburban = 1` for suburban homes, `0` if not, and then estimate the model. You could then do the same with `urban = 1` for urban homes, `0` if not. Each regression of these binomial Logistic regressions will give you the effect on the log-odds of someone buying a suburban or rural house respectively.

This is a valid approach, but since you fit 2 separate models you get separate fit statistics for each of the models. In addition, there are other problems associated with splitting the data this way. The multinomial Logistic model estimates the two binomial models together as a single model (or more binomial models for multiple categories) and give you one set of fit statistics for the whole model.

More generally, a multinomial model is one in which the outcome variable has K categorical values. In our example above K = 3. More generally, a multinomial Logistic model fits logistic regression composed of **K - 1** binomial Logistic models, but estimated together as a single model. There are R packages and functions that will estimate all K - 1 models jointly. The most popular ones are:

multinom(){nnet}, mlogit{mlogit}; and vglm{VGAM}**. I use the** vglm()** (Vector Generalized Linear Model) function in the **{VGAM}** (Vector Generalized and Additive Models) package in the examples below. This function runs multinomial Logistic and other categorical outcome regressions.

The model is fitted with the **Womenlf** (it's el-f, not one-f) data set in the **{car}** packages, which contains women labor force statistics.

```
library(VGAM) # Contains the vglm() function
library(car) # Contains the Womenlf (Women Labor Force) data set we use below

head(Womenlf) # Take a quick look at a few records

##     partic hincome children  region
## 1 not.work      15  present Ontario
## 2 not.work      13  present Ontario
## 3 not.work      45  present Ontario
## 4 not.work      23  present Ontario
## 5 not.work      19  present Ontario
## 6 not.work       7  present Ontario

# ? Womenlf # Use this command to explore the data set
```

Let's fit a model for labor participation (fulltime, not.work, parttime), based on husband's income in thousands and presence of children in the household (absence, presence). First, let's inspect the levels of the outcome variable **partic**.

```
levels(Womenlf$partic)

## [1] "fulltime" "not.work" "parttime"
```

- The levels in $partic are: **fulltime**, **not.work**, **parttime**
- **fulltime** is the refLevel, but we can always change this
- The effect for **not.work** will be suffixed **:1** in the summary output
- The effect for **parttime** will be suffixed **:2** in the summary output

```
vglm.fit <- vglm(partic ~ hincome + children,
                 family = multinomial(refLevel = 1), # "fulltime"
                 data = Womenlf)

summary(vglm.fit)

##
## Call:
## vglm(formula = partic ~ hincome + children, family = multinomial(refLevel
= 1),
##     data = Womenlf)
##
## Pearson residuals:
##                         Min      1Q  Median      3Q   Max
## log(mu[,2]/mu[,1]) -3.156 -0.7541  0.5405  0.6063 2.170
## log(mu[,3]/mu[,1]) -2.440 -0.2928 -0.2520 -0.1768 4.495
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept):1     -1.98282    0.48418  -4.095 4.22e-05 ***
## (Intercept):2     -3.41513    0.66552  -5.132 2.87e-07 ***
## hincome:1          0.09723    0.02810   3.461 0.000539 ***
## hincome:2          0.10412    0.03328   3.128 0.001759 **
## childrenpresent:1  2.55860    0.36220   7.064 1.62e-12 ***
## childrenpresent:2  2.58009    0.50972   5.062 4.15e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Names of linear predictors: log(mu[,2]/mu[,1]), log(mu[,3]/mu[,1])
##
## Residual deviance: 422.8819 on 520 degrees of freedom
##
## Log-likelihood: -211.441 on 520 degrees of freedom
##
## Number of Fisher scoring iterations: 5
##
## No Hauck-Donner effect found in any of the estimates
```

```
##
##
## Reference group is level  1  of the response
```

Interpretation:

The coefficients in the multinomial logistic regression, just like in binomial logit, show the effect of a 1 unit increase in the predictor variable on the log-odds of the dependent variable. But this is the difference:

- In binomial Logistic, the coefficient of a predictor represents, on average and holding everything else constant, the log-odds change in units of the outcome variable being 1 (relative to 0), when the predictor increases by 1. The odds effect represents the factor by which the respective odds change.

- In multinomial logit, the coefficient is about the log-odds of the response variable being in that category, relative to the reference category (i.e., the response variable left out of the one specified with the "refLevel" attribute)

- Fit Statistics, similar to binomial logistic with glm()

Let's start by looking at the fit statistics of the model. Notice that: the 2LL is 2 times the negative log-likelihood; 2LL and Deviance are the same thing; and AIC is like an **adjusted" 2LL (the more complex the model the larger the AIC, compared to the 2LL).

```
c("Log-Likelihood" = logLik(vglm.fit),
  "2LL" = -2 * logLik(vglm.fit),
  "Deviance" = deviance(vglm.fit),
  "AIC" = AIC(vglm.fit)) # Akaike Information Criterion

## Log-Likelihood            2LL       Deviance            AIC
##      -211.4410       422.8819       422.8819       434.8819
```

Let's now look at the effects or coefficients of the predictors. First, notice that each variable has 2 values. Since we have 3 categories in the response variable and the first one was left out as the reference variable, the first coefficient is for the log-odds of the second response category "not.work"; and the second one is for the third response category "parttime".

```
log.odds <- coef(vglm.fit)

round(
  cbind("Log-Odds" = log.odds,
      "Odds" = exp(log.odds)),
  digits = 3)

##                    Log-Odds   Odds
## (Intercept):1        -1.983  0.138
## (Intercept):2        -3.415  0.033
## hincome:1             0.097  1.102
## hincome:2             0.104  1.110
## childrenpresent:1    2.559 12.918
## childrenpresent:2    2.580 13.198
```

Interpretations:

hincome:1 - on average and holding everything else constant, when the husband's income goes up by $1K, the log-odds of **not working**, go up by 0.097, compared to working **full time**, The odds go up 1.102 times.

hincome:2 - On average and holding everything else constant, when the husband's income goes up by $1K, the log-odds of working **part time**, go up by 0.104. The odds go up 1.1097 times, compared to working **full time**.

childrenpresent:1 - on average and holding everything else constant, when there are children present in the family, compared to no children, the log-odds of **not working**, go up by 2.559, compared to working **full time**, The odds go up 12.917 times.

childrenpresent:2 - on average and holding everything else constant, when there are children present in the family, compared to no children, the log-odds of working **part time**, go up by 2.580, compared to working **full time**, The odds go up 13.198 times.

Let's now find predicted values with the **predictvglm()** function

```
pred.log.odds = predictvglm(vglm.fit, newdata = NULL, se.fit = T)

# Notes: newdata can be used to specify a test or new data set to predict if
omitted or NULL the predictions are done on the training set.
```

Just like with the coefficients, we get 2 predictions for each observation, one for the log-odds of being in the second category (compared to the first) and the other for the third category (compared to the first). Also, the **predictvglm()** (pred.log.odds) object has a few attributes. Let's list the first 10 values of each:

```
log.odds <- pred.log.odds$fitted.values # Predicted values, log odds
se <- pred.log.odds$se.fit # Standard errors
pred.odds <- exp(pred.log.odds$fitted.values) # Predicted odds
pred.prob <- pred.odds/(1+pred.odds) # Predicted probabilities

pred.all <- round(
               cbind(log.odds, se, pred.odds, pred.prob),
               digits = 3)

# vglm() reports the above data without column names, let's create them

colnames(pred.all) <- c("LogOdds2","LogOdds3",
                        "Std.Err2", "StdErr3",
                        "Odds2", "Odds3",
                        "Prob2", "Prob3")

pred.all[1:10, ] # Take a look at the first 10 rows
```

```
##    LogOdds2 LogOdds3 Std.Err2 StdErr3   Odds2  Odds3 Prob2 Prob3
## 1     2.034    0.727    0.259   0.297   7.646  2.068 0.884 0.674
## 2     1.840    0.519    0.246   0.290   6.295  1.680 0.863 0.627
## 3     4.951    3.850    0.958   1.102 141.338 47.016 0.993 0.979
## 4     2.812    1.560    0.394   0.440  16.644  4.758 0.943 0.826
## 5     2.423    1.143    0.314   0.351  11.281  3.137 0.919 0.758
## 6     1.256   -0.106    0.282   0.353   3.513  0.899 0.778 0.473
## 7     2.034    0.727    0.259   0.297   7.646  2.068 0.884 0.674
## 8     1.256   -0.106    0.282   0.353   3.513  0.899 0.778 0.473
## 9     2.034    0.727    0.259   0.297   7.646  2.068 0.884 0.674
## 10    2.812    1.560    0.394   0.440  16.644  4.758 0.943 0.826
```

6. Linear Discriminant Analysis (LDA)

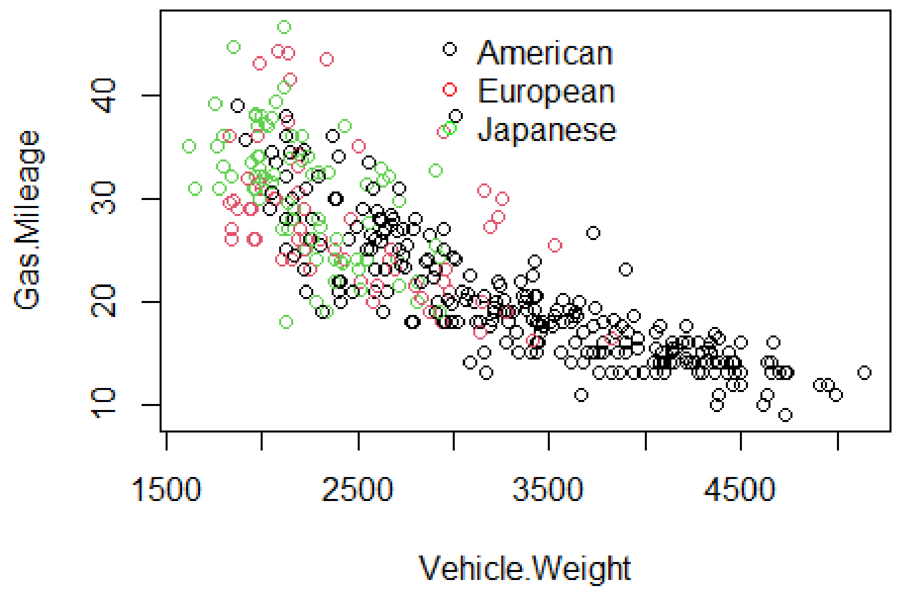
Let’s take a look at a few attributes of the **Auto** data set in the **{ISLR}** library.

```
library(ISLR)
options(scipen = 4)

# This is not needed for the analysis, just for an illustration plot:
# Auto$mpg <- ifelse(origin == 3, Auto$mpg * 1.2, Auto$mpg)

plot(Auto$mpg ~ Auto$weight,
     xlab = "Vehicle.Weight",
     ylab = "Gas.Mileage",
     col = Auto$origin)

legend(x = "top",
       c("American", "European", "Japanese"),
       col = c("black", "red", "green"),
       pch=1, bty = "n")
```



The diagram above suggests that if we know the weight and gas mileage of a vehicle, we may be able to predict the likelihood that the vehicle is American, European or Japanese. Let's first try a simple multinomial LDA model:

```
library(MASS) # Contains the lda() function
auto.fit.lda = lda(origin ~ weight + mpg, data = Auto)
auto.fit.lda$prior # Check out the prior probabilities for each class

##         1         2         3
## 0.6250000 0.1734694 0.2015306

auto.lda.pred <- predict(auto.fit.lda)

# Let's look at the first 10 posterior probabilities for each class
round(auto.lda.pred$posterior[1:10, ], digits = 4)

##         1      2      3
## 1  0.8964 0.0668 0.0367
## 2  0.9373 0.0434 0.0193
## 3  0.8855 0.0740 0.0405
## 4  0.9000 0.0679 0.0320
## 5  0.8953 0.0694 0.0353
## 6  0.9776 0.0153 0.0071
## 7  0.9796 0.0143 0.0061
## 8  0.9782 0.0153 0.0065
## 9  0.9818 0.0127 0.0055
## 10 0.9509 0.0339 0.0152
```

Let's now analyze a more complex data set with a binomial outcome. I use the **Heart.csv** data set provided by the ISLR package authors:

```
heart <- read.table("Heart.csv", sep=",", head=T)
head(heart)

##   sbp tobacco  ldl adiposity famhist typea obesity alcohol age chd
## 1 160   12.00 5.73     23.11 Present    49   25.30   97.20  52   1
## 2 144    0.01 4.41     28.61  Absent    55   28.87    2.06  63   1
## 3 118    0.08 3.48     32.28 Present    52   29.14    3.81  46   0
## 4 170    7.50 6.41     38.03 Present    51   31.99   24.26  58   1
## 5 134   13.60 3.50     27.78 Present    60   25.99   57.34  49   1
## 6 132    6.20 6.47     36.21 Present    62   30.77   14.14  45   0

# Note: I used the Heart.csv data file above, which I had already downloaded
from the ISLR website, but you could read the file directly from the web with
this command (commented out here):

# heart <-
read.table("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/SAheart.d
ata", sep = ",", head = T, row.names = 1)
```

We will use cross-validation to evaluate the accuracy of the model, so let's start by computing index vectors for the train and test sub-samples:

```
set.seed(1)
tr.size <- 0.7 # 70% train sub-sample
train <- sample(1:nrow(heart), tr.size * nrow(heart))
test <- seq(1:nrow(heart))[-train] # Any number not in the train vector
```

Let's now fit the model on the train data – i.e., train the LDA model

```
heart.fit.lda <- lda(chd ~ ., data = heart[train,])
heart.fit.lda # Check out the results

## Call:
## lda(chd ~ ., data = heart[train, ])
##
## Prior probabilities of groups:
##         0         1
## 0.6563467 0.3436533
##
## Group means:
##        sbp  tobacco      ldl adiposity famhistPresent    typea  obesity
## 0 134.9245 2.470283 4.373868  23.94382      0.2924528 51.98585 25.88882
## 1 142.9910 6.080450 5.625045  28.59550      0.6036036 53.74775 26.79865
##    alcohol      age
## 0 15.43590 38.57075
## 1 19.71351 50.62162
##
## Coefficients of linear discriminants:
```

```
##                              LD1
## sbp               0.00232257478
## tobacco           0.10180499306
## ldl               0.16755841467
## adiposity         0.01652028980
## famhistPresent  0.84833975602
## typea             0.02254757548
## obesity          -0.05952142157
## alcohol           0.00002954867
## age               0.02994590316
```

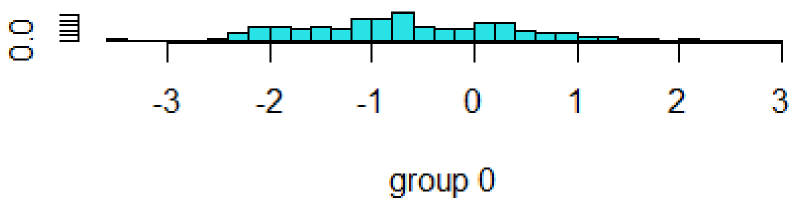
Note: the $scaling ** attributeofthis ** lda() ** object,** heart.fit.lda$**scaling** contains the linear discriminant function coefficients displayed above. Note that these are NOT regression coefficients, but rather the equation of the linear discriminant functio, which separates the outcome classifications into one category or another.

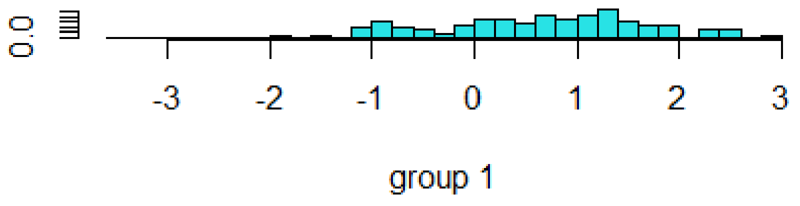
Another note: you can also use **heart.fit.lda$prior** to get the prior probabilities or proportions of the response variable.

```
# You can also get a summary, but it is not very useful
summary(heart.fit.lda)

##         Length Class  Mode
## prior    2     -none- numeric
## counts   2     -none- numeric
## means   18     -none- numeric
## scaling  9     -none- numeric
## lev      2     -none- character
## svd      1     -none- numeric
## N        1     -none- numeric
## call     3     -none- call
## terms    3     terms  call
## xlevels  1     -none- list

plot(heart.fit.lda) # Outcome distribution for each outcom class
```





Let's now extract the test subsample

```
heart.test <- heart[test, ] # Test sub-sample
```

Now let's make predictions with the train model and the test sub-sample

```
heart.lda.pred <- predict(heart.fit.lda, heart.test)
heart.lda.pred$posterior[1:10, ] # Inspect the respective probabilities

##              0          1
## 5  0.21907471 0.78092529
## 6  0.34229649 0.65770351
## 7  0.79750892 0.20249108
## 8  0.36714934 0.63285066
## 9  0.84219741 0.15780259
## 10 0.46317643 0.53682357
## 11 0.47794534 0.52205466
## 17 0.13991056 0.86008944
## 18 0.09529883 0.90470117
## 21 0.92396431 0.07603569

heart.lda.pred$class # Inspect the prdicted classifications for prob > 0.5

##   [1] 1 1 0 1 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0
1 1 1
##  [38] 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0
1 0 0
##  [75] 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
```

```
1 0 1
## [112] 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

Note: the display above shows the prob of **chd = 0** and of **chd = 1**. That's why we see 2 columns. The default classification threshold is **prob > 0.5**. That is, any observation with a **prob(chd = 1 > 0.5) will be predicted as 1, 0 otherwise. For most of what we do below, we only need the second column for** prob (chd = 1 > 0.5)**:

```
heart.lda.pred$posterior[1:10, 2] # Take a look

##          5          6          7          8          9         10
11
## 0.78092529 0.65770351 0.20249108 0.63285066 0.15780259 0.53682357
0.52205466
##         17         18         21
## 0.86008944 0.90470117 0.07603569
```

LDA Cross-Validated Confusion Matrix

Now that we have the trained model and some prediction objects, let's use them to construct the confusion matrix.

```
conf.mat <- table("Predicted" = heart.lda.pred$class,
                  "Actual"=heart.test$chd)

conf.mat

##          Actual
## Predicted  0  1
##         0 75 29
##         1 15 20
```

As we did before, let's compute accuracy the metrics and then add column and row totals and labels (as I mentioned earlier, if we name the cross-table conf.mat, then all the scripts we used earlier for the Logistic regression confusion matrix will work here without change)

```
TruN   <- conf.mat[1,1];  TruP <- conf.mat[2,2]
FalN   <- conf.mat[1,2];  FalP <- conf.mat[2,1]
TotN   <- TruN + FalP;    TotP <- TruP + FalN
TotNpr <- TruN + FalN;  TotPpr <- TruP + FalP
Tot    <- TotN + TotP

# Quick check of the computations
cbind(TruN, TruP, FalN, FalP, TotN, TotP, TotNpr, TotPpr, Tot)

##      TruN TruP FalN FalP TotN TotP TotNpr TotPpr Tot
## [1,]   75   20   29   15   90   49    104     35 139
```

Let's add totals and labels and display the confusion matrix

```
conf.mat.totals <- cbind(conf.mat, c(TotNpr, TotPpr))
conf.mat.totals <- rbind(conf.mat.totals, c(TotN, TotP, Tot))

colnames(conf.mat.totals) <-
  rownames(conf.mat.totals) <-
     c("No","Yes", "Total")

conf.mat.totals # Display the confusion matrix

##       No Yes Total
## No    75  29   104
## Yes   15  20    35
## Total 90  49   139
```

Now let's use these metrics to compute accuracy and error rates

```
Accuracy.Rate <- (TruN + TruP) / Tot
Error.Rate <- (FalN + FalP) / Tot
Sensitivity <- TruP / TotP # Proportion of correct positives
Specificity <- TruN / TotN # Proportion of correct negatives
FalP.Rate <- 1 - Specificity # Proportion of false positives

# Let's combine them into one vector:

lda.rates.50 <- c(Accuracy.Rate, Error.Rate, Sensitivity, Specificity,
FalP.Rate)

# Let's name the vector elements:

names(lda.rates.50) <- c("Accuracy", "Rate",
                         "Sensitivity", "Specificity", "False Pos")

lda.rates.50 # Display the results

##    Accuracy        Rate Sensitivity Specificity   False Pos
##   0.6834532   0.3165468   0.4081633   0.8333333   0.1666667
```

We can do the same with other values of lambda, say 0.60. We can use the data in the probability outcome vector we computed earlier to change the classification threshold to, say **prob(chd = 1 > 0.60). Notice that we use the** heart.lda.pred$posterior** vector we computed before. The probability outcomes did not change. We are only changing the threshold for when to classify an outcome as a 0 or a 1.

```
lambda <- 0.6

heart.lda.class.60 <- ifelse(heart.lda.pred$posterior[,2] > lambda, 1, 0)
heart.lda.class.60 [1:20] # Take a look at a few revised classifications

##  5  6  7  8  9 10 11 17 18 21 30 32 34 46 47 52 54 55 57 58
##  1  1  0  1  0  0  0  1  1  0  0  0  0  0  1  0  0  0  0  0
```

Now, let's build the confusion matrix

```
# Cross-table

conf.mat <- table(heart.lda.class.60, heart.test$chd)

TruN <- conf.mat[1,1];  TruP <- conf.mat[2,2]
FalN <- conf.mat[1,2];  FalP <- conf.mat[2,1]
TotN <- TruN + FalP;    TotP <- TruP + FalN
TotNpr <- TruN + FalN;  TotPpr <- TruP + FalP
Tot <- TotN + TotP

# Quick check of the computations
cbind(TruN, TruP, FalN, FalP, TotN, TotP, TotNpr, TotPpr, Tot)

##      TruN TruP FalN FalP TotN TotP TotNpr TotPpr Tot
## [1,]   77   17   32   13   90   49    109     30 139
```

Let's add totals and labels and display the confusion matrix

```
conf.mat.totals <- cbind(conf.mat, c(TotNpr, TotPpr))
conf.mat.totals <- rbind(conf.mat.totals, c(TotN, TotP, Tot))

colnames(conf.mat.totals) <-
  rownames(conf.mat.totals) <-
     c("No","Yes", "Total")

conf.mat.totals # Display the confusion matrix

##       No Yes Total
## No    77  32   109
## Yes   13  17    30
## Total 90  49   139
```

Now, the accuracy statistics:

```
Accuracy.Rate <- (TruN + TruP) / Tot
Error.Rate <- (FalN + FalP) / Tot
Sensitivity <- TruP / TotP # Proportion of correct positives
Specificity <- TruN / TotN # Proportion of correct negatives
FalP.Rate <- 1 - Specificity # False Positive Rate

lda.rates.60 <- c(Accuracy.Rate, Error.Rate, Sensitivity, Specificity,
FalP.Rate)

names(lda.rates.60) <- c("Accuracy", "Error",
                         "Sensitivity", "Specificity", "False Pos")

round(lda.rates.60, digits = 3)
```

```
##    Accuracy       Error Sensitivity Specificity   False Pos
##       0.676       0.324       0.347       0.856       0.144
```

Now, let's display the accuracy statistics with both thresholds together, so that we can compare results.

```
round(rbind(lda.rates.50, lda.rates.60), digits = 3)

##              Accuracy  Rate Sensitivity Specificity False Pos
## lda.rates.50    0.683 0.317       0.408       0.833     0.167
## lda.rates.60    0.676 0.324       0.347       0.856     0.144
```

Let's include the Binary Logistic accuracy metrics

```
round(rbind(logit.rates.50, logit.rates.60,
            lda.rates.50, lda.rates.60),
      digits = 3)

##                Accuracy Error Sensitivity Specificity False Pos
## logit.rates.50    0.698 0.302       0.469       0.822     0.178
## logit.rates.60    0.676 0.324       0.327       0.867     0.133
## lda.rates.50      0.683 0.317       0.408       0.833     0.167
## lda.rates.60      0.676 0.324       0.347       0.856     0.144
```

Which method is best overall? Which method is better for predicting positives? And for negatives? I think you have all the tools, information and wisdom to answer these questions now.

ROC Curve and LDA

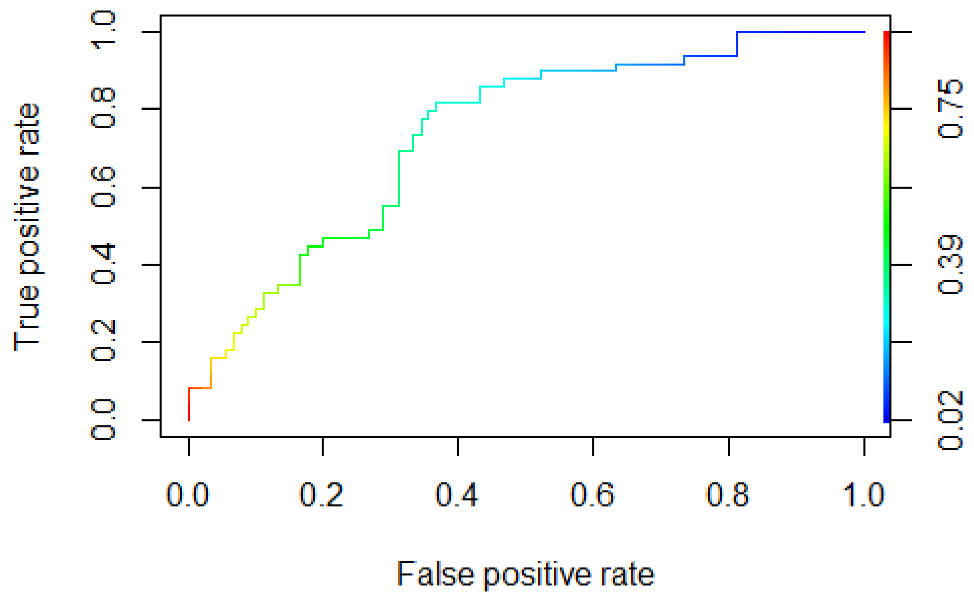
We can construct the ROC curve for the LDA model in the exact same way we did for the Logistic regression model.

```
library(ROCR)

pred <- prediction(heart.lda.pred$posterior[,2],
                   heart$chd[test])

perf <- performance(pred,"tpr","fpr")

plot(perf, colorize=TRUE)
```



```r
auc=performance(pred,"auc") # Compute the AUC
c(auc@y.name[[1]], auc@y.values[[1]]) # Display the AUC

## [1] "Area under the ROC curve" "0.734013605442177"
```

7. Quadratic Discriminant Analysis (QDA)

The syntax for **QDA** identical to **LDA**, except that we use the **qda()** function instead of **lda()**. Both functions are in the **{MASS}** library. LDA and QDA fullfill similar goals and work in similar ways, except that LDA uses linear discriminant functions, whereas QDA uses quadratic discriminant functions. Which one is better? It all depends on the data and how it is distributed across the predictors. The best thing to do is to try both and evaluate them with cross-validation testing, as illustrated below.

```r
library(MASS) # Contains the qda(){MASS} function

heart <- read.table("Heart.csv", sep=",", head=T)
```

Compute index vectors for train and test sub-samples, as shown for LDA above. Then fit the QDA model using the **{MASS}qda()** function.

```r
heart.fit.qda <- qda(chd ~ ., data = heart[train,])

# Inspect the results
```

```
heart.fit.qda # See the linear discriminant function
```

```
## Call:
## qda(chd ~ ., data = heart[train, ])
##
## Prior probabilities of groups:
##         0         1
## 0.6563467 0.3436533
##
## Group means:
##        sbp  tobacco      ldl adiposity famhistPresent    typea  obesity
## 0 134.9245 2.470283 4.373868  23.94382      0.2924528 51.98585 25.88882
## 1 142.9910 6.080450 5.625045  28.59550      0.6036036 53.74775 26.79865
##    alcohol      age
## 0 15.43590 38.57075
## 1 19.71351 50.62162
```

```
summary(heart.fit.qda) # Not very useful
```

```
##         Length Class  Mode
## prior     2    -none- numeric
## counts    2    -none- numeric
## means    18    -none- numeric
## scaling 162    -none- numeric
## ldet      2    -none- numeric
## lev       2    -none- character
## N         1    -none- numeric
## call      3    -none- call
## terms     3    terms  call
## xlevels   1    -none- list
```

Let's extract the test sub-sample and make probability predictions

```
heart.test <- heart[test, ]
heart.qda.pred <- predict(heart.fit.qda, heart.test)
```

Now, let's take a look at some of the attributes of the **heart.qda.pred** object

```
heart.qda.pred$class # Inspect the classifications
```

```
##   [1] 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0
1 1 1
##  [38] 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0
0 0 0
##  [75] 0 1 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
1 0 1
## [112] 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1

# Check out the first 10 records of the posterior probabilities > 0.5 that
chd is 0 and 1. Or just the probabilities for chd = 1 (second column only)
```

```
heart.qda.pred$posterior[1:10, ] # Both probabilities

##              0           1
## 5  0.06173275 0.938267248
## 6  0.31774112 0.682258877
## 7  0.92202810 0.077971899
## 8  0.82553199 0.174468010
## 9  0.98542350 0.014576499
## 10 0.29010239 0.709897612
## 11 0.18939482 0.810605179
## 17 0.01297368 0.987026318
## 18 0.06682875 0.933171250
## 21 0.99428720 0.005712804

heart.qda.pred$posterior[1:10, 2] # Just prob(chd = 1)

##           5           6           7           8           9          10
## 0.938267248 0.682258877 0.077971899 0.174468010 0.014576499 0.709897612
##          11          17          18          21
## 0.810605179 0.987026318 0.933171250 0.005712804

# Notice how any prediction probability of 0.5 for chd = 1 got a predicted
classification of 1, and 0 otherwise.
```

Now let's work on the confusion matrix

```
conf.mat <- table("Predicted"=heart.qda.pred$class,
                  "Actual"=heart$chd[test])

TruN <- conf.mat[1,1];  TruP <- conf.mat[2,2]
FalN <- conf.mat[1,2];  FalP <- conf.mat[2,1]
TotN <- TruN + FalP;    TotP <- TruP + FalN
TotNpr <- TruN + FalN;  TotPpr <- TruP + FalP
Tot <- TotN + TotP

# Quick check of the computations
cbind(TruN, TruP, FalN, FalP, TotN, TotP, TotNpr, TotPpr, Tot)

##      TruN TruP FalN FalP TotN TotP TotNpr TotPpr Tot
## [1,]   70   21   28   20   90   49     98     41 139
```

Let's add totals and labels and display the confusion matrix

```
conf.mat.totals <- cbind(conf.mat, c(TotNpr, TotPpr))
conf.mat.totals <- rbind(conf.mat.totals, c(TotN, TotP, Tot))

colnames(conf.mat.totals) <-
  rownames(conf.mat.totals) <-
     c("No","Yes", "Total")
```

```
conf.mat.totals # Display the confusion matrix

##        No Yes Total
## No     70  28    98
## Yes    20  21    41
## Total  90  49   139
```

Now, the accuracy statistics:

```
Accuracy.Rate <- (TruN + TruP) / Tot
Error.Rate <- (FalN + FalP) / Tot
Sensitivity <- TruP / TotP # Proportion of correct positives
Specificity <- TruN / TotN # Proportion of correct negatives
FalP.Rate <- 1 - Specificity # False Positive Rate

qda.rates.50 <- c(Accuracy.Rate, Error.Rate,
                  Sensitivity, Specificity, FalP.Rate)

names(qda.rates.50) <- c("Accuracy Rate", "Error Rate",
                         "Sensitivity", "Specificity", "False Positives")

round(rbind(qda.rates.50), digits = 3)

##              Accuracy Rate Error Rate Sensitivity Specificity False
Positives
## qda.rates.50         0.655      0.345       0.429       0.778
0.222
```

As we did with LDA, we can use the data in the QDA probability outcome vector we computed earlier to change the classification threshold to, say **prob(chd = 1 > 0.60)**.

```
lambda <- 0.6

heart.qda.class.60 <- ifelse(heart.qda.pred$posterior[, 2] > lambda, 1, 0)

# Cross-table

conf.mat <- table(heart.qda.class.60, heart.test$chd)

TruN <- conf.mat[1,1];  TruP <- conf.mat[2,2]
FalN <- conf.mat[1,2];  FalP <- conf.mat[2,1]
TotN <- TruN + FalP;    TotP <- TruP + FalN
TotNpr <- TruN + FalN;  TotPpr <- TruP + FalP
Tot <- TotN + TotP

# Quick check of the computations
cbind(TruN, TruP, FalN, FalP, TotN, TotP, TotNpr, TotPpr, Tot)
```

```
##      TruN TruP FalN FalP TotN TotP TotNpr TotPpr Tot
## [1,]   74   20   29   16   90   49    103     36 139
```

Let’s add totals and labels and display the confusion matrix

```
conf.mat.totals <- cbind(conf.mat, c(TotNpr, TotPpr))
conf.mat.totals <- rbind(conf.mat.totals, c(TotN, TotP, Tot))

colnames(conf.mat.totals) <-
  rownames(conf.mat.totals) <-
     c("No","Yes", "Total")

conf.mat.totals # Display the confusion matrix

##       No Yes Total
## No    74  29   103
## Yes   16  20    36
## Total 90  49   139
```

Now, the accuracy statistics:

```
Accuracy.Rate <- (TruN + TruP) / Tot
Error.Rate <- (FalN + FalP) / Tot
Sensitivity <- TruP / TotP # Proportion of correct positives
Specificity <- TruN / TotN # Proportion of correct negatives
FalP.Rate <- 1 - Specificity # False Positive Rate

qda.rates.60 <- c(Accuracy.Rate, Error.Rate, Sensitivity, Specificity,
FalP.Rate)

names(qda.rates.60) <- c("Accuracy", "Error",
                         "Sensitivity", "Specificity", "False Pos")

round(qda.rates.60, digits = 3)

##    Accuracy       Error Sensitivity Specificity   False Pos
##       0.676       0.324       0.408       0.822       0.178
```

Now, let’s display the accuracy statistics for borh, LDA and QDA, with both thresholds together, 0.5 and 0.6, so that we can compare results.

Let’s include the Binary Logistic accuracy metrics

```
round(rbind(logit.rates.50, logit.rates.60,
            lda.rates.50, lda.rates.60,
            qda.rates.50, qda.rates.60),
      digits = 3)

##                Accuracy Error Sensitivity Specificity False Pos
## logit.rates.50    0.698 0.302       0.469       0.822     0.178
## logit.rates.60    0.676 0.324       0.327       0.867     0.133
## lda.rates.50      0.683 0.317       0.408       0.833     0.167
## lda.rates.60      0.676 0.324       0.347       0.856     0.144
## qda.rates.50      0.655 0.345       0.429       0.778     0.222
## qda.rates.60      0.676 0.324       0.408       0.822     0.178
```

Which method is best overall? Which method is better for predicting positives? And for negatives? I think you have all the tools, information and wisdom to answer these questions now.