Multivariate classification vignette

Logistic regression, LDA, QDA

Oscar Trevizo

2023-05-24

Contents

This code and functions are based on lessons from Harvard Statistical Learning class [see references]. I expanded the material with my own scripts, notes and R documentation and I plan to continue adding examples overtime.

These scripts focus on a simulated multivariate dataset. A multivariate problem differs slightly from the problem with only one explanatory variable.

1 Load the libraries

```
# library(dplyr)
# library(tidyr)
library(ggplot2)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
# library(ggExtra)

library(stats)                # Stats contains glm for logistic regression
library(MASS)                 # LDA and QDA
library(caret)                # Performance function
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.2.3
```

```
library(pROC)                 # ROC
```

```
## Warning: package 'pROC' was built under R version 4.2.3
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

2 Functions

Adapted from R functions shared by faculty in Harvard data science class (2021). See references at the bottom of this notebook.

```
###
#
# prediction.metrics function -- to return a list with all the metrics values
#
# Based on R functions shared by faculty in Harvard data science class (2021). See references.
#
# Input: truth and predicted lists.
#
# Returns a list with:
# [1] OBS = Observations or truth cases
# [2] Accuracy. ACC = sum(truth == predicted) * 100/length(truth)
# [3] Sensitivity. TPR True Positive Rate = TP/(TP + FN) = TP/P
# [4] Specificity. TNR True Negative Rate = TN/(FP + TN) = TN/N
# [5] Precision. Positive Predictive Value. PPV = TP/(TP + FP)
# [6] Negative Predictive Value. NPV = TN/(TN + FN)
# [7] False Discovery Rate. FDR = FP/(TP + FP)
# [8] False Positive Rate. FPR = FP/(FP + TN) = FP/N
# [9] True Positives. TP = sum(truth == 1 & predicted == 1)
# [10] True Negatives. TN = sum(truth == 0 & predicted == 0)
# [11] False Positives. FP = sum(truth == 0 & predicted == 1)
# [12] False Negatives. FN = sum(truth == 1 & predicted == 0)
# [13] Positives. P = TP + FN  # total number positives in the truth data
# [14] Negatives. N = FP + TN  # total number of negatives
#
prediction.metrics = function(truth, predicted) {
    # same length:
    if (length(truth) != length(predicted)) {
        stop("truth and predicted must be same length!")
    }
    # check for missing values (we are going to compute metrics on non-missing
    # values only)
    bKeep = !is.na(truth) & !is.na(predicted)
    predicted = predicted[bKeep]
    truth = truth[bKeep]
    # only 0 and 1:
    if (sum(truth %in% c(0, 1)) + sum(predicted %in% c(0, 1)) != 2 * length(truth)) {
        stop("only zeroes and ones are allowed!")
    }
    # how predictions align against known training/testing outcomes: TP/FP=
    # true/false positives, TN/FN=true/false negatives
```

```r
    TP = sum(truth == 1 & predicted == 1)
    TN = sum(truth == 0 & predicted == 0)
    FP = sum(truth == 0 & predicted == 1)
    FN = sum(truth == 1 & predicted == 0)
    P = TP + FN  # total number of positives in the truth data
    N = FP + TN  # total number of negatives
    # Add the following output to return (OAT 11/9/2021)
    OBS = length(truth)
    ACC = sum(truth == predicted)/length(truth)
    TPR = TP/P
    TNR = TN/N
    PPV = TP/(TP + FP)
    NPV = TN/(TN + FN)
    FDR = FP/(TP + FP)
    FPR = FP/N

    # Returned a named list
    output <- list(OBS=OBS, ACC=ACC, TPR=TPR, TNR=TNR, PPV=PPV,
                   NPV=NPV, FDR=FDR, FPR=FPR, TP=TP,
                   TN=TN, FP=FP, FN=FN, P=P, N=N)
    return(output)
}

print.the.metrics = function(metrics){
  cat(' OBS = ', metrics$OBS, '..................number of observations')
  cat('\n ACC = ', metrics$ACC, '.................Accuracy')
  cat('\n TPR = ', metrics$TPR, '.................True Positive Rate')
  cat('\n TNR = ', metrics$TNR, '.................True Negative Rate')
  cat('\n PPV = ', metrics$PPV, '.................Positive Predictive Value (Precision)')
  cat('\n NPV = ', metrics$NPV, '.................Negative Predictive Value')
  cat('\n FDR = ', metrics$FDR, '.................False Discover Rate')
  cat('\n FPR = ', metrics$FPR, '.................False Positive Rate')
  cat('\n TP  = ', metrics$FP, '.................True Positives')
  cat('\n TN  = ', metrics$TN, '.................True Negatives')
  cat('\n FP  = ', metrics$TN, '.................False Positives')
  cat('\n FN  = ', metrics$FN, '.................False Negatives')
  cat('\n P   = ', metrics$P, '.................Positives')
  cat('\n N   = ', metrics$N, '.................Negatives')


}
# Logistic regression
lgr.pred.ftn = function(formula, df.train, df.test){
  glm.fit <- glm(formula, data = df.train, family = binomial)
  glm.probs <- predict(glm.fit, newdata = df.test, type = "response")
  glm.pred <- rep(0, dim(df.test)[1])
  glm.pred[glm.probs>0.5]=1
  return(glm.pred)
}

# Linear Discriminant Analysis (LDA)
lda.pred.ftn = function(formula, df.train, df.test){
  lda.fit <- lda(formula, data = df.train)
```

```
  lda.pred <- predict(lda.fit, df.test)
  lda.class <- lda.pred$class
  return(lda.class)
}

# Quadratic Discriminant Analysis (QDA)
qda.pred.ftn = function(formula, df.train, df.test){
  qda.fit <- qda(formula, data = df.train)
  qda.pred <- predict(qda.fit, df.test)
  qda.class <- qda.pred$class
  return(qda.class)
}
```

3 Simulate the data

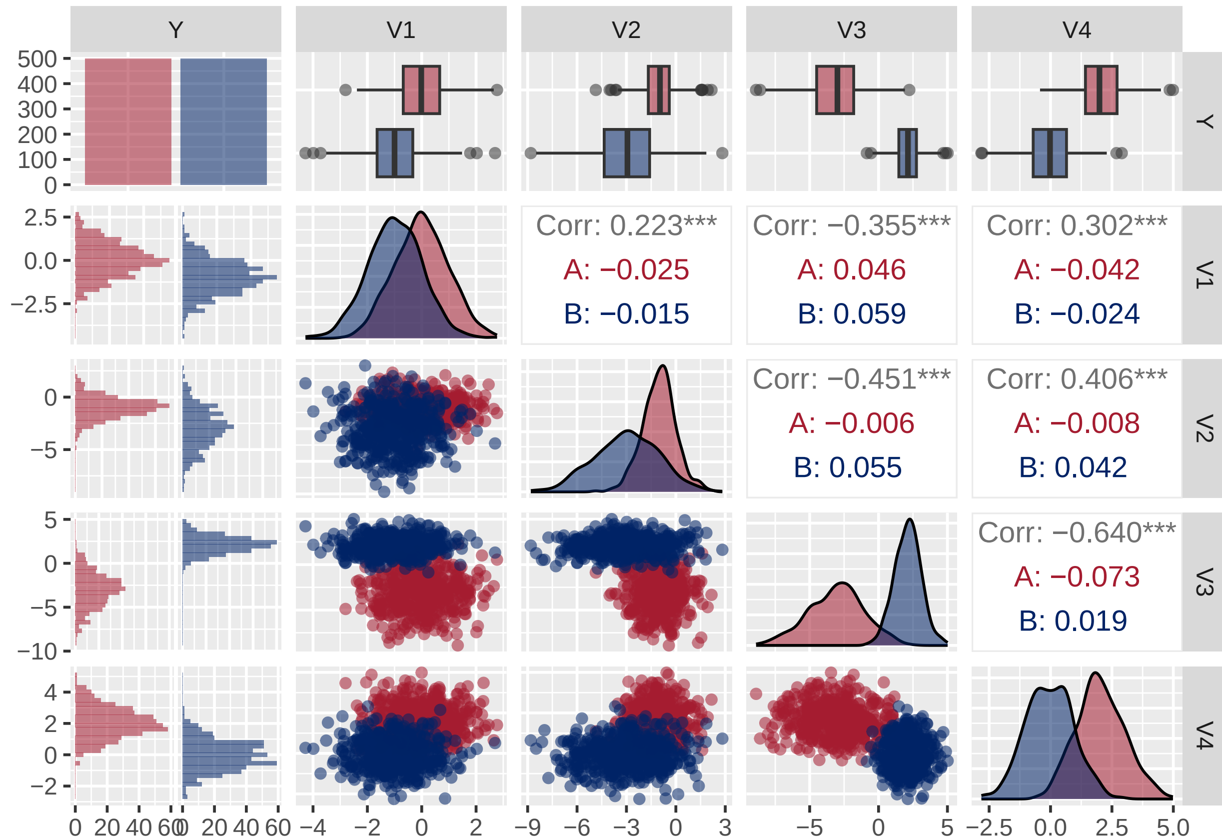
Play with two sets of Normally distributed sets of data with different means. We can change the number of samples and we can move the means around.

```
# From Harvard data science class (see references at the end of this notebook)

set.seed(11)

N = 1000
mu_1 = 0
mu_2 = -1
mu_3 = -3
mu_4 = 2

# Our measuring variable is continuous, numeric...
# ...it has two Normal distribution waves
# The first N observations has 0 mu, the next two variables have different mu
# Break each in half and mix the mu's around to simulate the data
v1 <- c(rnorm(N/2, mean=mu_1, sd = 1), rnorm(N/2, mean=mu_2, sd = 1))
v2 <- c(rnorm(N/2, mean=mu_2, sd = 1), rnorm(N/2, mean=mu_3, sd = 2))
v3 <- c(rnorm(N/2, mean=mu_3, sd = 2), rnorm(N/2, mean=mu_4, sd = 1))
v4 <- c(rnorm(N/2, mean=mu_4, sd = 1), rnorm(N/2, mean=mu_1, sd = 1))

# Our outcome is categorical, A and B xxxx times each
# ...the idea is to match A and B to a number x
# We want to break it in half; i.e. half for A and half for B
y <- rep(c("A", "B"), each=N/2)

# I sued this commented code in my univariate vignette.
# Make a data.frame with 1 and 0 values for Y
# The first column is Y the second column is X
# df <- data.frame(Y=ifelse(y=="A",0, 1), X=x)

# Here I will make Y a factor from the start.
# Either method works. Thought it was good to show the R factor way here.
# Be careful of how R assigns the values... It will do 1 and 2 instead of 0 and 1
```

```r
df <- data.frame(Y = factor(y),
                 V1 = v1, V2 = v2, V3 = v3, V4 = v4)
```

4 Pair plots

Pairplots are a better choice than trying to do individual boxplots and histogram in this case.



5 Build train and test sets

```r
set.seed(12321)

# Method #1, my method
# Get 2:1 random sample ratio for Train:Test sets
# sampleTrain <- sample(c(TRUE,FALSE,TRUE), nrow(df), rep=TRUE)
# df.train <- df[sampleTrain,]
# df.test <- df[!sampleTrain,]

# Method #2, traditional
# Traditionally we would split the df up and down as follows
ind <- sample(2, nrow(df), replace = TRUE, prob = c(0.7, 0.3))
df.train <- df[ind == 1,]
df.test <- df[ind == 2,]
```

6 Logistic regression (Generalized Linear Models GLM)

Needs library{stats}

6.1 Fit the model

```
##
#
# GLA from library{stats}
#
##
glm.fit <- glm(Y~., data=df.train, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Y ~ ., family = binomial, data = df.train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.10340  -0.00101   0.00007   0.01375   1.84139
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.1698     0.7209  -0.235   0.8138
## V1           -1.0141     0.4486  -2.260   0.0238 *
## V2           -0.6740     0.2979  -2.263   0.0236 *
## V3            3.0080     0.5493   5.476 4.36e-08 ***
## V4           -2.6152     0.5819  -4.494 6.98e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 923.25  on 665  degrees of freedom
## Residual deviance:  50.32  on 661  degrees of freedom
## AIC: 60.32
##
## Number of Fisher Scoring iterations: 10
```

6.2 Predict

- Make predictions using the test dataset.

```r
##
#
# predict{stats}
#
# Continued based on ISLR 4.6.2 p.156-158
#

glm.probs <- predict(glm.fit, newdata = df.test, type = "response")

# Per ISLR, we need contrasts() and use the variable as a logical vector.
# Note, I already had converted Room as.factor
# contrasts(df$Y)

# Initiated glm.pred vector
glm.pred = rep(0, dim(df.test)[1])
# Adjust the probability. Here is something one can play with after looking at the 'table' that follows
glm.pred[glm.probs>0.5]=1
```

6.3 Confusion matrix

```r
##
#
# Continued based on ISLR 4.6.2 p.156-158
#
# Numbers outside of the diagonal are either false positives or false negatives
#

table(glm.pred, df.test$Y)
```

```
##
## glm.pred   A   B
##        0 167   1
##        1   2 164
```

```r
mean(glm.pred == df.test$Y)
```

```
## [1] 0
```

6.4 Prediction metrics (function)

- Now I will use the function calculate accuracy, sensitivity, and specificity

```r
##
#
# Based on functions from above
#

lgr.pred <- lgr.pred.ftn(Y~., df.train, df.test)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Here we want numeric, not factors... just how the function is built
lgr.metrics <- prediction.metrics(ifelse(df.test$Y == 'A', 0, 1), lgr.pred)

print.the.metrics(lgr.metrics)
```

```
##  OBS =  334 ..................number of observations
##  ACC =  0.991018 .................Accuracy
##  TPR =  0.9939394 .................True Positive Rate
##  TNR =  0.9881657 .................True Negative Rate
##  PPV =  0.9879518 .................Positive Predictive Value (Precision)
##  NPV =  0.9940476 .................Negative Predictive Value
##  FDR =  0.01204819 .................False Discover Rate
##  FPR =  0.01183432 .................False Positive Rate
##  TP  =  2 .................True Positives
##  TN  =  167 .................True Negatives
##  FP  =  167 .................False Positives
##  FN  =  1 .................False Negatives
##  P   =  165 .................Positives
##  N   =  169 .................Negatives
```

6.5 Prediction performance {carat}

```r
confusionMatrix(as.factor(lgr.pred),
                factor(ifelse(df.test$Y=="A",0, 1)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 167   1
##          1   2 164
##
##                Accuracy : 0.991
##                  95% CI : (0.974, 0.9981)
##     No Information Rate : 0.506
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.982
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9882
##             Specificity : 0.9939
##          Pos Pred Value : 0.9940
##          Neg Pred Value : 0.9880
##              Prevalence : 0.5060
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5030
##       Balanced Accuracy : 0.9911
```

```
##
##        'Positive' Class : 0
##
```

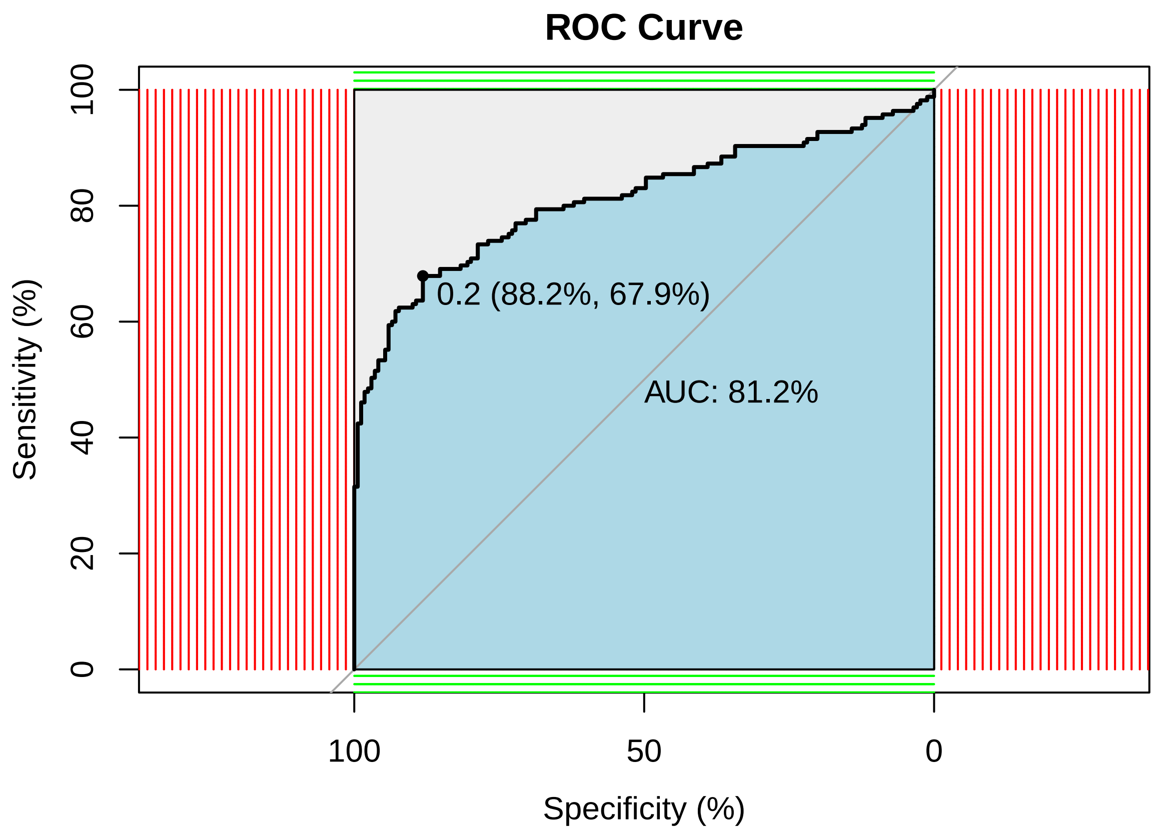
6.6 ROC

WARNING: There is a bug here currently.

```
#### ROC
# p1 <- predict(glm.fit, df.test, type = 'terms')
# p1 <- p1[,2]
# r <- multiclass.roc(df.test$Y, p1, percent = TRUE)
# roc <- r[['rocs']]
# r1 <- roc[[1]]
# plot.roc(r1,
#          print.auc=TRUE,
#          auc.polygon=TRUE,
#          grid=c(0.1, 0.2),
#          grid.col=c("green", "red"),
#          max.auc.polygon=TRUE,
#          auc.polygon.col="lightblue",
#          print.thres=TRUE,
#          main= 'ROC Curve')

# re-mdel but use 'terms'
p1 <- predict(glm.fit, newdata = df.test, type = 'terms')
p1 <- p1[,2]
r <- roc(df.test$Y, p1, percent = TRUE)
```

```
## Setting levels: control = A, case = B

## Setting direction: controls < cases
```

```
plot.roc(r,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```




```
AUC <- as.numeric(r[['auc']])
```

7 Linear Discriminant Analysis (LDA)

Needs library{MASS}

7.1 Fit the model

```
##
#
# LDA from library{MASS}
#
##
lda.fit <- lda(Y~., data = df.train)
summary(lda.fit)
```

```
##         Length Class  Mode
## prior   2      -none- numeric
## counts  2      -none- numeric
## means   8      -none- numeric
## scaling 4      -none- numeric
## lev     2      -none- character
```

```
## svd      1      -none- numeric
## N        1      -none- numeric
## call     3      -none- call
## terms    3      terms  call
## xlevels 0      -none- list
```

7.2 Predict

```
lda.pred <- predict(lda.fit, df.test)

names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

7.3 Confusion matrix

```
##
#
# Continued based on ISLR 4.6.3 p.161-162
#
#

lda.class <- lda.pred$class

table(lda.class, df.test$Y)
```

```
##
## lda.class   A   B
##         A 167   0
##         B   2 165
```

```
mean(lda.class == df.test$Y)
```

```
## [1] 0.994012
```

- The confusion matrix is based on the test set.
- The confusion matrix indicates the number of observations correctly predicted not to be in Y.
- And it indicated the number of observations correctly predicted to be in Y.
- The `mean()` function calculates the diagonals over the total.
- These results parallel those from linear regression in Problem 1.

7.4 Prediction metrics (function)

- Now I will use the function from from above

```r
##
#
# Based on functions from above
#

lda.pred <- lda.pred.ftn(Y~., df.train, df.test)

# Here we want numeric, not factors... just how the function is built
lda.metrics <- prediction.metrics(ifelse(df.test$Y == 'A', 0, 1), lgr.pred)

print.the.metrics(lda.metrics)
```

```
##  OBS =  334 ..................number of observations
##  ACC =  0.991018 .................Accuracy
##  TPR =  0.9939394 .................True Positive Rate
##  TNR =  0.9881657 .................True Negative Rate
##  PPV =  0.9879518 .................Positive Predictive Value (Precision)
##  NPV =  0.9940476 .................Negative Predictive Value
##  FDR =  0.01204819 .................False Discover Rate
##  FPR =  0.01183432 .................False Positive Rate
##  TP   =  2 .................True Positives
##  TN   =  167 .................True Negatives
##  FP   =  167 .................False Positives
##  FN   =  1 .................False Negatives
##  P    =  165 .................Positives
##  N    =  169 .................Negatives
```

7.5 Prediction performance {carat}

```r
# The factor here is handled differently than in the lgr case above
confusionMatrix(as.factor(lda.pred), df.test$Y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B
##          A 167   0
##          B   2 165
##
##                Accuracy : 0.994
##                  95% CI : (0.9785, 0.9993)
##     No Information Rate : 0.506
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.988
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.9882
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
```

```
##          Neg Pred Value : 0.9880
##              Prevalence : 0.5060
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9941
##
##        'Positive' Class : A
##
```

7.6 ROC

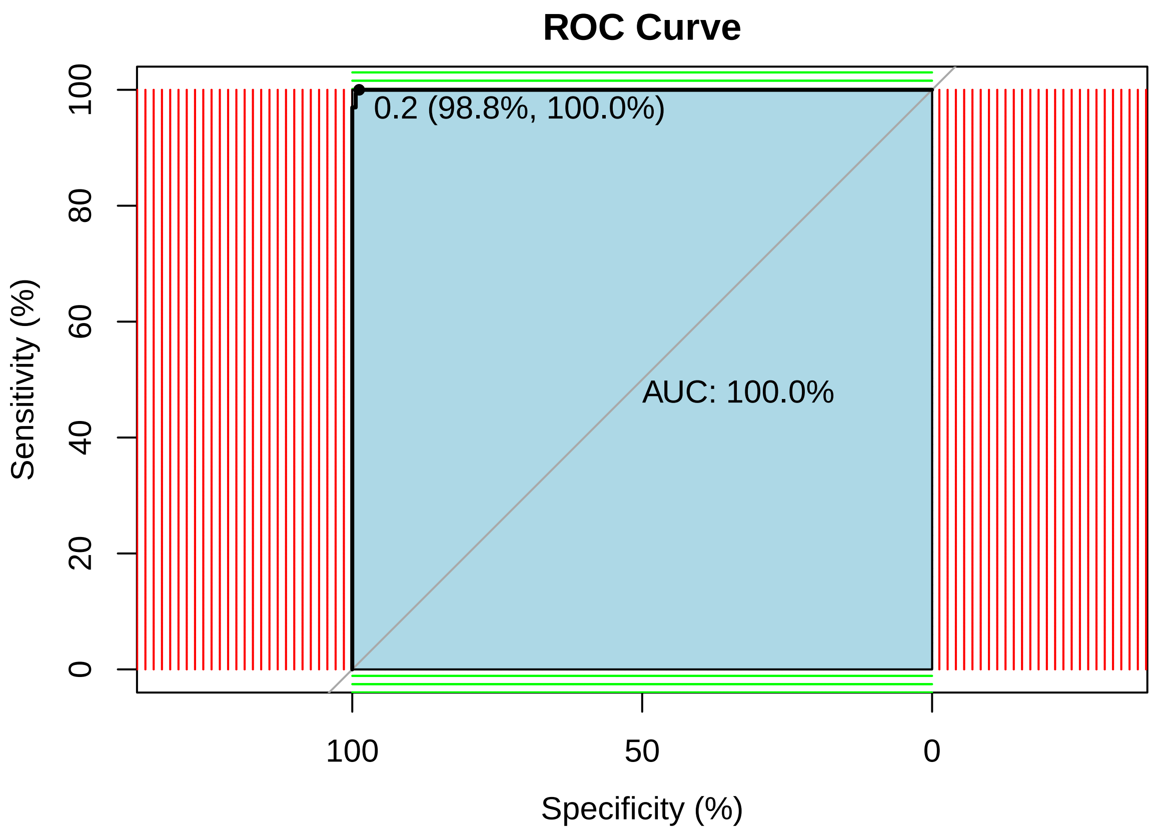
```r
p1 <- predict(lda.fit, newdata = df.test, type = 'terms')
r <- roc(df.test$Y, p1$x, percent = TRUE)
```

```
## Setting levels: control = A, case = B
```

```
## Warning in roc.default(df.test$Y, p1$x, percent = TRUE): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

```
## Setting direction: controls < cases
```

```r
plot.roc(r,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```




```
AUC <- as.numeric(r[['auc']])
```

8 Quadratic Discriminant Analysis (QDA)

Needs library{MASS}

8.1 Fit the model

```
##
#
# QDA from library{MASS}
#
##
qda.fit <- qda(Y~., data = df.train)
summary(qda.fit)
```

```
##         Length Class  Mode
## prior    2     -none- numeric
## counts   2     -none- numeric
## means    8     -none- numeric
## scaling 32     -none- numeric
## ldet     2     -none- numeric
```

```
## lev      2     -none- character
## N        1     -none- numeric
## call     3     -none- call
## terms    3     terms  call
## xlevels  0     -none- list
```

8.2 Predict

- Make predictions using the test dataset.

```
##
#
# predict{stats}
#
# Continued based on ISLR 4.6.4 p.163
#

qda.pred <- predict(qda.fit, df.test)
```

8.3 Confusion matrix

```
##
#
# Continued based on ISLR 4.6.4 p.163
#
#

qda.class <- qda.pred$class

table(qda.class, df.test$Y)
```

```
##
## qda.class   A   B
##         A 167   0
##         B   2 165
```

```
mean(qda.class == df.test$Y)
```

```
## [1] 0.994012
```

8.4 Prediction metrics (function)

- Now I will use the function calculate accuracy, sensitivity, and specificity

```
##
#
# Based on functions from above
#
```

```r
qda.pred <- qda.pred.ftn(Y~., df.train, df.test)

# Here we want numeric, not factors... just how the function is built
qda.metrics <- prediction.metrics(ifelse(df.test$Y == 'A', 0, 1), lgr.pred)

print.the.metrics(qda.metrics)
```

```
##  OBS =  334 ...................number of observations
##  ACC =  0.991018 .................Accuracy
##  TPR =  0.9939394 .................True Positive Rate
##  TNR =  0.9881657 .................True Negative Rate
##  PPV =  0.9879518 .................Positive Predictive Value (Precision)
##  NPV =  0.9940476 .................Negative Predictive Value
##  FDR =  0.01204819 .................False Discover Rate
##  FPR =  0.01183432 .................False Positive Rate
##  TP  =  2 ..................True Positives
##  TN  =  167 ..................True Negatives
##  FP  =  167 ..................False Positives
##  FN  =  1 ..................False Negatives
##  P   =  165 ..................Positives
##  N   =  169 ..................Negatives
```

8.5 Prediction performance {carat}

```r
confusionMatrix(factor(qda.pred), df.test$Y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B
##          A 167   0
##          B   2 165
##
##                Accuracy : 0.994
##                  95% CI : (0.9785, 0.9993)
##     No Information Rate : 0.506
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.988
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.9882
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9880
##              Prevalence : 0.5060
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9941
```

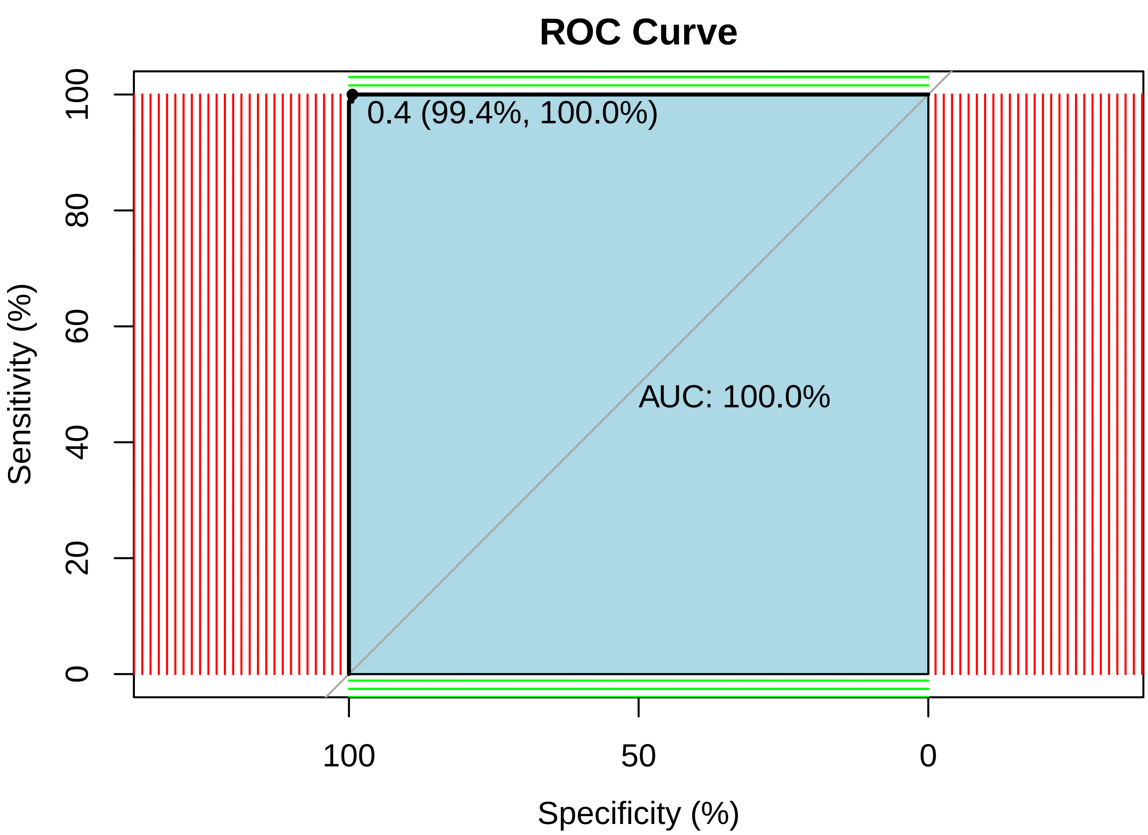
```
##
##        'Positive' Class : A
##
```

8.6 ROC

```
p1 <- predict(qda.fit, newdata = df.test, type = 'terms')
r <- roc(df.test$Y, p1$posterior[,1], percent = TRUE)
```

```
## Setting levels: control = A, case = B

## Setting direction: controls > cases
```

```
plot.roc(r,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```



```r
AUC <- as.numeric(r[['auc']])
```

9 References

- Harvard "Elements of Statistical Learning" (2021) taught by professors Dr. Sivachenko, Dr. Farutin
- Book "An Introduction to Statistical Learning with Applications in R" (ISLR) by Gareth James et al