**Chapter 20: Proper ROC models Online Appendices**

# Table of contents

Online Appendix 20.A: Viewing slopes and ROC curves

Online Appendix 20.B: Plotting PROPROC ROC curves

Online Appendix 20.C: Plotting CBM ROC curves

Online Appendix 20.D: Plotting bigamma model curves

# Appendix 20.A: Viewing slopes and ROC curves

The following code generates an ROC plot for specified values of the binormal *a*, *b* parameters. It was used to generate book Figure 20.4.1 (a - d).

## Appendix 20.A.1 Code listing: mainImproperRocs.R

# mainImproperRocs.R

rm( list = ls())

require(ggplot2)

source("rocY.R")

aArray <- c(0.7, 0.7, 1.5, 2)

bArray <- c(0.5, 1.5, 0.5, 0.5)

z <- seq(-3, 5, by = 0.01)

FPF <- seq(0.0, 1, 0.01)

for (i in 1:length(aArray))

{

a <- aArray[i]

b <- bArray[i]

TPF <- rocY(FPF, a, b)

rocPlot <- data.frame(FPF = FPF, TPF = TPF)

p <- ggplot(rocPlot, aes(x = FPF, y = TPF)) +

geom\_line(size = 2) +

scale\_color\_manual(values = "black") +

theme(axis.title.y = element\_text(size = 25,face="bold"),

axis.title.x = element\_text(size = 30,face="bold")) +

scale\_x\_continuous(expand = c(0, 0)) +

scale\_y\_continuous(expand = c(0, 0))

print(p)

cat("a = ", a, ", b = ", b, "\n")

}

This is fairly straightforward code that the reader should be able to understand using standard debugging tools, such as inserting a break point and clicking Source followed by Next. The function rocY() returns the binormal model predicted ordinate corresponding to a specified FPF and model parameters. Source the code to obtain the cited figures.

## Appendix 20.A.2 Code listing: mainPdfs.R

The following code, in file mainPdf.R generates plots of the ROC curve and pdfs for . The code in file mainPdf1.R is identical except for a different plotting routine that produces grey-scale instead of color plots, as the book figures are limited to grey-scale; it was used to generate book Figure 20.2 (a - b).

# mainPdfs.R

rm( list = ls())

require(ggplot2)

source("rocY.R")

aArray <- c(0.7, 0.7, 1.5, 2)

bArray <- c(0.5, 1.5, 0.5, 0.5)

z1 <- seq(-5, 3, by = 0.01)

z2 <- seq(-5, 7, by = 0.01)

FPF <- seq(0.0, 1, 0.01)

pdf1 <- dnorm(z1)

for (i in 1:length(aArray))

{

a <- aArray[i]

b <- bArray[i]

TPF <- rocY(FPF, a, b)

rocPlot <- data.frame(FPF = FPF, TPF = TPF)

plotRoc <- ggplot(

rocPlot, aes(x = FPF, y = TPF)) +

geom\_line()

print(plotRoc)

pdf2 <- dnorm(z2, a/b, sd = 1/b)

df <- data.frame(

z = c(z1, z2), pdfs = c(pdf1, pdf2),

truth = c(rep('non-diseased', length(pdf1)),

rep('diseased', length(pdf2))))

rocpdfs <- ggplot(

df,

aes(x = z, y = pdfs, color = truth)) +

geom\_line()

print(rocpdfs)

cat("a = ", a, ", b = ", b, "\n")

break

}

Again, this is fairly straightforward code which is left to the reader to explore. Line 12 defines pdf1, the pdf of the decision variable for non-diseased cases, i.e., . Line 25 defines pdf2, the pdf of the decision variable for diseased cases, i.e., . See book chapter 6 for the meanings of the binormal model parameters. Source the code to reveal the plots. Notice the wide spread of the diseased pdf causing the diseased pdf value to exceed that of the non-diseased pdf at both very high and very low values of z, leading to the "hook" at the upper right corner of the ROC plot.

## Appendix 20.A.3 Code listing: mainDegenerate.R

The following code generated book Figure 20.3 (a - d).

rm(list = ls()) # mainDegenerate.R

source("BMCurve.R")

source("CBMCurve.R")

plotOP <- data.frame(FPF = 0, TPF = 0.75)

a <- 0.6744898; b <- 0

plotCurve <- BMCurve(a, b)

figA <- ggplot(mapping = aes(x = FPF, y = TPF)) +

geom\_line(data = plotCurve) +

geom\_point(data = plotOP)

print(figA)

a <- 1.281552; b <- 0

plotCurve <- BMCurve(a, b)

figB <- ggplot(mapping = aes(x = FPF, y = TPF)) +

geom\_line(data = plotCurve) +

geom\_point(data = plotOP)

print(figB)

a <- Inf; b <- 0

plotCurve <- BMCurve(a, b)

figC <- ggplot(mapping = aes(x = FPF, y = TPF)) +

geom\_line(data = plotCurve) +

geom\_point(data = plotOP)

print(figC)

mu <- Inf; alpha <- 0.75

plotCurve <- CBMCurve(mu, alpha)

figD <- ggplot(mapping = aes(x = FPF, y = TPF)) +

geom\_line(data = plotCurve) +

geom\_point(data = plotOP)

print(figD)

File BMCurve.R, sourced at line 2, has the function BMCurve(a, b) corresponding to the binormal model (abbreviated to BM) predicted (smooth) curve. The corresponding CBM function CBMCurve(mu, alpha) is sourced at line 3. Line 5 defines the single operating point plotOP (0,0.75) as a data frame object. Note that the single operating point on the y-axis defines a degenerate dataset. Line 8 calls BMPoints()with binormal model parameters a = 0.6744898 and b = 0. Returned is a list object plotCurve, which contains FPF and TPF. Insert a break point at line 14 and click Source. The following code snippet shows the "smooth" curve, actually consisting of three line segments, Figure 1.

Browse[2]> str(plotCurve)

'data.frame': 4003 obs. of 2 variables:

$ FPF: num 1 1 1 1 1 1 1 1 1 1 ...

$ TPF: num 1 0.75 0.75 0.75 0.75 ...

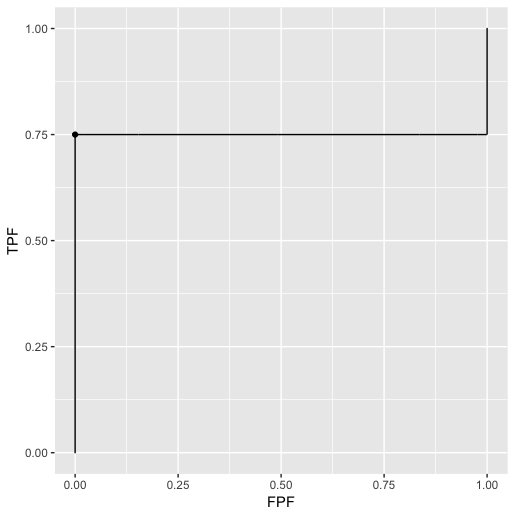


Figure : Result of sourcing code with break point at line 14.

The figures are named figA, etc., to correspond to the book, one of the better examples of the author's documentation style. Note that R allows Inf as a value, lines 21 and 28. See book section 20.5.1 for further details. Removing the break point and sourcing the code yields the cited figures in the book.

## Appendix 20.A.4 Code listing: mainSlopes.R

The following code generated book Figure 20.4 (a - f).

# mainSlopes.R

rm( list = ls())

require(ggplot2)

# values used in Fig. 20.4 (A-F) of book

a <- 0.7;b <- 0.5

zArray <- list(seq(-5, 8, by = 0.01),

seq(-3, 5, by = 0.01),

seq(-3, 3, by = 0.01),

seq(-2, 2, by = 0.01),

seq(-1, 1, by = 0.01),

seq(-1, 0, by = 0.01))

for (i in 1:length(zArray))

{

z <- zArray[[i]]

slope <-b\*dnorm(a-b\*z)/dnorm(-z) # same as likelihood ratio

slopePlot <- data.frame(z = z, slope = slope)

plotSlope <- ggplot(slopePlot, aes(x = z, y = slope)) +

geom\_line(size = 2) +

theme(axis.title.y = element\_text(size = 25,face="bold"),

axis.title.x = element\_text(size = 30,face="bold")) +

scale\_x\_continuous(expand = c(0, 0)) +

scale\_y\_continuous(expand = c(0, 0))

print(plotSlope)

cat("a = ", a, ", b = ", b, "\n")

}

Line 17 defines slope (an array, since z is an array; confirm by inserting a break point at line 18 and sourcing the code) using the formula for the likelihood ratio corresponding to the binormal model; book Equation 20.5. The following lines plot the slope as a function of z, the binormal model decision variable. See book Section 20.6 for further details on the meaning of these plots.

# Appendix 20.B: Plotting PROPROC ROC curves

The following code generates Fig. 20.5 (a – d). The relevant book section is 20.7.1.

## Appendix 20.B.1 Code listing: mainProprocRocs.R

# mainProprocRocs.R

rm( list = ls())

require(caTools)

require(ggplot2)

source("rocY.R")

source("proprocFunctions.R")

c1Arr <- c(-0.1322804, 0.2225588)

daArr <- c(1.197239,1.740157)

for (i in 1:2)

{

c1 <- c1Arr[i]

da <- daArr[i]

ret <- Transform2ab(da, c1)

a <- ret$a;b <- ret$b

# may need to adjust limits to view detail of slope plot

if (i == 1) z <- seq(-3, 0, by = 0.01)

if (i == 2) z <- seq(-3, 5, by = 0.01)

FPF <- seq(0.0, 1, 0.001)

TPF <- rocY(FPF, a, b)

rocPlot <- data.frame(FPF = FPF, TPF = TPF)

plotRoc <- ggplot(rocPlot, aes(x = FPF, y = TPF)) +

geom\_line()

print(plotRoc)

slope <-b\*dnorm(a-b\*z)/dnorm(-z) # same as likelihood ratio

slopePlot <- data.frame(z = z, slope = slope)

plotSlope <- ggplot(slopePlot, aes(x = z, y = slope)) +

geom\_line()

print(plotSlope)

cat("\nc = ", c1, "\nda = ", da,"\n")

}

Line 6 sources proprocFunctions.R, which codes the relevant formulae from book Section 20.7. The reader should inspect this file. One of its functions, Transform2ab(), is used at line 14 to convert PROPROC parameters (*c, da*) to binormal parameters (*a,b*). Line 28 calculates the slope of the binormal model ROC. The computation is done for two pairs of PROPROC parameters from a clinical dataset, the values are coded at line 8-9 and printed upon sourcing the code.

> source('~/onlinebookk21778/Ch20/software/mainProprocRocs.R')

c = -0.1322804

da = 1.197239

c = 0.2225588

da = 1.740157

See the cited book section for the meanings of the ROC and slope plots.

## Appendix 20.B.2: PROPROC parameters for the Tony dataset

The .lrc format ROC ratings data (this is inferred ROC data created from the original FROC data contained in TonyData.xlsx using capability of RJafroc) is in file TonyData.lrc. One needs to run DBM-MRMC software available from either of two websites (e.g., do a search for "University of Iowa ROC software"). Select PROPROC for the fitting method and AUC for the figure of merit. The program generates four output files with names, as follows, appended to "TonyData proproc": (1) "area jackknife obuch anova.txt", (2) "area plot.csv", (3) "area pooled.csv" and (4) "area pseudo.csv". Open the 3rd file under RStudio. Its contents are listed below:

## Appendix 20.B.3 Contents of "...area pooled.CSV"

T,R,returnCode,area,numCAT,adjPMean,c,d\_a,

1, 1, 0, 0.8014164671, 6, 0.8014164671, -0.1322803605, 1.1972392948

1, 2, 0, 0.8947898031, 5, 0.8947898031, -0.0869651348, 1.7711756372

1, 3, 0, 0.8526605379, 6, 0.8526605379, -0.1444418518, 1.4819348764

1, 4, 0, 0.8577776011, 6, 0.8577776011, 0.0804601608, 1.5137569144

1, 5, 0, 0.8909392545, 6, 0.8909392545, 0.2225587648, 1.7401572166

2, 1, 0, 0.6716574327, 6, 0.6716574327, -0.0817424757, 0.6281251326

2, 2, 0, 0.7544738610, 5, 0.7544738610, 0.0497644849, 0.9738785560

2, 3, 0, 0.7931787103, 6, 0.7931787103, -0.1326126225, 1.1558706614

2, 4, 0, 0.8740273581, 6, 0.8740273581, 0.1182226331, 1.6201757157

2, 5, 0, 0.7360982768, 6, 0.7360982768, 0.0781032985, 0.8928815903

The last two columns list the appropriate  values. The values in the last two columns were copied to a file "TonyData.R" and used to create the arrays c1[5,2] and da[5,2] and this file is sourced, at line 6, in file mainMetzPanEqn36Check.R. This method of data entry, not generally recommended, was done to preserve "vertical real-estate" in the listing of the main code that follows:

## Appendix 20.B.4: Code listing mainMetzPanEqn36Check

# mainMetzPanEqn36Check.R

rm( list = ls())

require(ggplot2)

require(caTools)

require(mvtnorm)

source("proprocFunctions.R")

source("TonyData.R") # this contains the saved values from the PROPROC run

npts <- 10000

for (i in 1:2) {

for (j in 1:5) {

C <- c1[i,j]

da <- d\_a1[i,j]

ret <- GetLimits(da,C)

LL <- ret$LL;UL <- ret$UL

vc <- seq (LL, UL, length.out = npts)

TPF <- TruePositiveFraction (vc, da, C)

FPF <- FalsePositiveFraction (vc, da, C)

FPF <- rev(FPF);TPF <- rev(TPF)

df2 <- data.frame(FPF = FPF, TPF = TPF)

plotRoc <- ggplot(df2, aes(x = FPF, y = TPF)) + geom\_line()

# print(plotRoc) not to be overwhelmed with plots

# do integral numerically

numAuc <- trapz(FPF, TPF)

# Implement Eqn. 36 from Metz-Pan paper

rho <- -(1-C^2)/(1+C^2);sigma <- rbind(c(1, rho), c(rho, 1))

lower <- rep(-Inf,2);upper <- c(-da/sqrt(2),0)

A\_prop <- pnorm(da/sqrt(2)) + 2 \* pmvnorm(lower, upper, sigma = sigma)

A\_prop <- as.numeric(A\_prop)

cat("i = ", i,"j = ", j,"C = ", C, ", da = ", da, "NumericalAUC = ", numAuc, ", Eqn. 36 = ", A\_prop,"\n")

}

}

Line 22 is commented so that we are not overwhelmed with 10 plots. The function GetLimits(da,C) at line 13 implements Eqn. 20.17 for the appropriate limits of v (called, in deference to Metz's original notation, vc in the code). Notice that the author used upper case C for the parameter, as lower case c is the concatenation function as in c(1,2). Line 16 creates an array starting from LL (lower limit) to UL (upper limit) of length npts = 10,000. A large number of steps were used so that the simple trapezoidal AUC, used at line 25, closely approximates true AUC. Line 17-18 implements book Eqn. 20.14 and Eqn. 20.15 for FPF and TPF, respectively, and line 19 reverses the ordering of these arrays (otherwise [1,1] values are (1,1), instead of (0,0), and the trapezoidal area comes out negative). Lines 27-31 implements Eqn. 20.18 for the proper AUC. The final line prints out the modality and reader numbers, the parameter values and both the numerical estimate of AUC and the analytic expression for AUC. Source the file:

> source('~/onlinebookk21778/Ch20/software/mainMetzPanEqn36Check.R')

i = 1 j = 1 C = -0.1322804 , da = 1.197239 NumericalAUC = 0.8014164 , Eqn. 36 = 0.8014164

i = 1 j = 2 C = -0.08696513 , da = 1.771176 NumericalAUC = 0.8947898 , Eqn. 36 = 0.8947898

i = 1 j = 3 C = -0.1444419 , da = 1.481935 NumericalAUC = 0.8526604 , Eqn. 36 = 0.8526605

i = 1 j = 4 C = 0.08046016 , da = 1.513757 NumericalAUC = 0.8577776 , Eqn. 36 = 0.8577776

i = 1 j = 5 C = 0.2225588 , da = 1.740157 NumericalAUC = 0.8909392 , Eqn. 36 = 0.8909392

i = 2 j = 1 C = -0.08174248 , da = 0.6281251 NumericalAUC = 0.6716573 , Eqn. 36 = 0.6716574

i = 2 j = 2 C = 0.04976448 , da = 0.9738786 NumericalAUC = 0.7544738 , Eqn. 36 = 0.7544739

i = 2 j = 3 C = -0.1326126 , da = 1.155871 NumericalAUC = 0.7931786 , Eqn. 36 = 0.7931787

i = 2 j = 4 C = 0.1182226 , da = 1.620176 NumericalAUC = 0.8740273 , Eqn. 36 = 0.8740274

i = 2 j = 5 C = 0.0781033 , da = 0.8928816 NumericalAUC = 0.7360989 , Eqn. 36 = 0.7360989

Note the close correspondence between the numerical values and those predicted by Metz-Pan Eqn. 36. These results are summarized in book Table 20.2.

# Appendix 20.C: Plotting CBM ROC curves

## Appendix 20.C.1 Code Listing: mainCbmPlots.R

rm(list = ls()) #mainCbmPlots.R

library(ggplot2)

source("cbmFunctions.R")

FPF <- seq(0.0, 1, 0.001)

alphaArr <- c(0.2, 0.8);muArr <- c(1,3)

for (i in 1:2)

for (j in 1:2)

{

{

alpha <- alphaArr[i]

mu <- muArr[j]

TPF <- CbmRocY(FPF, mu, alpha)

rocPlot <- data.frame(FPF = FPF, TPF = TPF)

plotRoc <- ggplot(rocPlot, aes(x = FPF, y = TPF)) + geom\_line() +

geom\_line(data = rocPlot)

print(plotRoc)

if (i == 1) {

z1 <- seq(-3, 3, by = 0.01)

z2 <- seq(-3, mu + 3, by = 0.01)

} else {

z1 <- seq(-3, 3, by = 0.01)

z2 <- seq(-3, mu + 3, by = 0.01)

}

Pdf1 <- dnorm(z1)

Pdf2 <- (1 - alpha) \* dnorm(z2) + alpha \* dnorm(z2, mu)

df <- data.frame(

z = c(z1, z2), pdf = c(Pdf1, Pdf2),

truth = c(rep('non-diseased', length(Pdf1)),

rep('diseased', length(Pdf2)))

)

cbmPdfs <- ggplot(df, aes(x = z, y = pdf, color = truth)) +

geom\_line(data = df)

print(cbmPdfs)

cat("mu = ", mu, ", alpha = ", alpha, "\n")

next

}

}

For convenience the contents of cbmFuntions.R is reproduced below:

CbmRocY <- function (x, mu, alpha) {

y <- (1-alpha)\*(1-pnorm(qnorm(1-x))) + alpha\*(1-pnorm(qnorm(1-x)-mu))

return(y)

}

The real action happens in line 14. The function CbmRocY(), defined in cbmFunctions.R, takes as first argument x, an array containing the FPF values at which to evaluate TPF. Since x (i.e., FPF) is the area (array) *above* a threshold, 1-x is the area (or probability) *below* the threshold, which is a CDF or function; the qnorm function (which is the inverse of the CDF) applied to it yields the cutoff  array corresponding to x. With this explanation, and the fact that pnorm is the function, one should be able to see the correspondence between Eqn. 20.28 and line 2 in cbmFunctions.R. Source mainCbmPlots.R to get Fig. 20.7 (a - d). As before, mainCbmPlots1.R generates the corresponding grey-scale plots (and larger font sizes) for the book figures.

# Appendix 20.D: Plotting bigamma model curves

The code is in file mainBigammaPlots.R.

## Appendix 20.D.1 Code Listing

rm(list = ls()) #mainBigammaPlots.R

require(stats)

require(ggplot2)

aucIntegrand <- function (FPF, r, lambda)

{

y <- 1 - pgamma(qgamma(1-FPF, r), r, scale = 1/lambda)

return(y)

}

biGammaRocY <- function (xi, r, lambda) {

y <- 1 - pgamma(xi, r, scale = 1/lambda)

return(y)

}

rArray <- c(1,4.391,5,10);lambdaArray <- c(1,0.439,0.3,0.1)

#r <- 1; lambda <- 1 # I made it up

#r <- 4.391; lambda <- 0.439 # from Dorfman paper

#r <- 5; lambda <- 0.3 # I made it up

#r <- 10; lambda <- 0.1 # I made it up

# left limit below gets the upper end of the ROC curve; the right limit gets the lower corner

for (i in 1:length(rArray))

{

r <- rArray[i];lambda <- lambdaArray[i]

xi <- seq(0, r/lambda + 10\*sqrt(r/lambda^2), by = 0.01)

FPF <- biGammaRocY(xi, r, 1)

TPF <- biGammaRocY(xi, r, lambda)

rocPlot <- data.frame(FPF = FPF, TPF = TPF)

plotRoc <- ggplot(rocPlot, aes(x = FPF, y = TPF)) +

geom\_line() +

geom\_line(data = rocPlot)

print(plotRoc)

xi <- seq(0.01, r/lambda + 3\*sqrt(r/lambda^2), by = 0.01) # taking mean + 3 times the standard deviation

Pdf1 <- dgamma(xi, r)

Pdf2 <- dgamma(xi, r, scale = 1/lambda)

df <- data.frame(xi = c(xi, xi), pdf = c(Pdf1, Pdf2),

truth = c(rep('non-diseased', length(Pdf1)),

rep('diseased', length(Pdf2))))

bigammapdfs <- ggplot(df, aes(x = xi, y = pdf, color = truth)) +

geom\_line()

print(bigammapdfs)

AUC <- integrate(aucIntegrand,0,1, r = r, lambda = lambda)

cat("r = ", r, ", lambda = ", lambda, ", AUC = ", AUC$value, "\n")

}

**Explanation**: Two functions are defined in this code. Since they are not used anywhere else, why assign them to separate files? Skipping the 1st function to line 12 - 15, this uses the CDF of the gamma distribution, namely pgamma(zi,r,scale=1/lambda), which is the integral from 0 to  (note my creative use of the variable name "zi" for the symbol) under the gamma *pdf* with shape parameter *r* and scale parameter . The complement of this integral is FPF provided , Eqn. 20.39, and TPF provided , Eqn. 20.40. Line 19, currently commented out, has the values of the parameters from the Dorfman et al publication. Line 17 specifies the values of the parameters in two arrays and the actual values are extracted inside the for-loop at line 25. Line 36 specifies a zi array (the author had to experiment with the upper limit to capture the entire ROC plot; currently it is set for the mean of the diseased distribution plus 10 times the standard deviation of the diseased distribution.) Lines 27 - 28 calculate the corresponding FPF and TPF arrays. Note the use of lambda = 1 in line 27. Lines 30 - 34 plots the ROC curve. Lines 38 – 47 plots the *pdf* functions. Line 49 integrates to get AUC under the ROC from FPF = 0 to FPF = 1. The function to be integrated, aucIntegrand <- function (FPF, r, lambda), is defined at lines 5 – 9. The function uses the quantile function qgamma to get from FPF to zi, i.e., qgamma(1-FPF, r), and then supplies this value of zi to pgamma with lambda defined appropriately, and returns, as always, the complement of the probability. [Note R's consistent naming of the distributions and the quantile functions: pnorm, qnorm, pgamma, qgamma, etc.]