**Title of relevant chapter: Hypothesis Testing**

# Online Appendix 7.A: Hypothesis Testing Single-Modality Single-Reader

The following code relates to book Section 8.2.

### Online Appendix 8.A.1: Code Listing

rm(list = ls()) #mainHT1R1M.R

source("Wilcoxon.R")

seed <- 1;set.seed(seed)

mu <- 1.5;sigma <- 1.3;K1 <- 50;K2 <- 52

# cheat to find the population mean and std. dev.

AUC <- array(dim = 10000)

for (i in 1:length(AUC)) {

zk1 <- rnorm(K1)

zk2 <- rnorm(K2, mean = mu, sd = sigma)

AUC[i] <- Wilcoxon(zk1, zk2)

}

meanAUC <- mean(AUC);sigmaAUC <- sd(AUC)

cat("pop mean AUC = ", meanAUC,

"\npop sigma AUC = ", sigmaAUC, "\n")

# one more trial, this is the one we want

# to compare to meanAUC,

zk1 <- rnorm(K1);zk2 <- rnorm(K2, mean = mu, sd = sigma)

AUC <- Wilcoxon(zk1, zk2)

cat("New AUC = ", AUC, "\n")

z <- (AUC - meanAUC)/sigmaAUC

#z <- qnorm(0.05/2)

cat("z-statistic = ", z, "\n")

# p value for two-sided AH

p2tailed <- pnorm(-abs(z)) + (1-pnorm(abs(z)))

# p value for one-sided AH > 0

p1tailedGT <- 1-pnorm(z)

# p value for one-sided AH < 0

p1tailedLT <- pnorm(z)

alpha <- 0.05

# critical value for two-sided AH:

# AUC not equal to meanAUC

z2tailed <- -qnorm(alpha/2)

# critical value for one-sided AH:

# AUC > meanAUC

z1tailedGT <- qnorm(1-alpha)

# critical value for one-sided AH:

# AUC < meanAUC

z1tailedLT <- qnorm(alpha)

cat("alpha of test = ", alpha, "\n")

cat("\nTwo-sided AH: AUC not equal to meanAUC", "\n")

cat("Critical value for two-sided AH:", z2tailed, "\n")

cat("p value for two-sided AH:", p2tailed, "\n")

cat("\nOne-sided AH: AUC > meanAUC", "\n")

cat("Critical value for one-sided AH:", z1tailedGT, "\n")

cat("p value for two-sided AH:", p1tailedGT, "\n")

cat("\nOne-sided AH: AUC < meanAUC", "\n")

cat("Critical value for one-sided AH:", z1tailedLT, "\n")

cat("p value for two-sided AH:", p1tailedLT, "\n")

Line 5 sets the parameters of the binormal simulation model. Line 8 – 16 "cheats", i.e., or uses "brute-force" method to find the mean and standard deviation of the population AUC. Since empirical AUC is being used it would be incorrect to use formula, book Section 6.4.4, for the standard deviation of the parametric area under the binormal ROC curve. Line 20 generates a new dataset and line 21 calculates the corresponding empirical AUC. Line 24 calculates the z-statistic, which is distributed N(0,1), book Equation 8.3. Line 29 calculates, book Equation 8.4, the p-value for the two-sided alternative hypothesis. Line 31 calculates the p-value for the one-sided alternative hypothesis, where large values lead to rejection of the NH, while line 33 calculates the p-value for the one-sided alternative hypothesis, where small values lead to rejection of the NH. Lines 38, 41 and 43 calculate the corresponding critical values of the z-statistic. The remaining statements print values.

### Online Appendix 8.A.2: Code Output

> source(...)

pop mean AUC = 0.819178

pop sigma AUC = 0.04176683

New AUC = 0.8626923

z-statistic = 1.04184

alpha of test = 0.05

Two-sided AH: AUC not equal to meanAUC

Critical value for two-sided AH: 1.959964

p value for two-sided AH: 0.297486

One-sided AH: AUC > meanAUC

Critical value for one-sided AH: 1.644854

p value for two-sided AH: 0.148743

One-sided AH: AUC < meanAUC

Critical value for one-sided AH: -1.644854

p value for two-sided AH: 0.851257

None of the tests reject the null hypothesis. The p-values of one-sided tests are half that for the two-sided test.

# Online Appendix 8.B: Type I errors

The following code relates to book Section 8.3.

### Online Appendix 8.B.1: Code Listing

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

source("Wilcoxon.R")

seed <- 1;set.seed(seed)

mu <- 1.5;sigma <- 1.3;K1 <- 50;K2 <- 52

# cheat to find the population mean and std. dev.

AUC <- array(dim = 10000)

for (i in 1:length(AUC)) {

zk1 <- rnorm(K1);zk2 <- rnorm(K2, mean = mu, sd = sigma)

AUC[i] <- Wilcoxon(zk1, zk2)

}

sigmaAUC <- sqrt(var(AUC));muAUC <- mean(AUC)

nTrials <- 2000

alpha <- 0.05 # size of test

reject = array(0, dim = nTrials)

for (trial in 1:length(reject)) {

zk1 <- rnorm(K1);zk2 <- rnorm(K2, mean = mu, sd = sigma)

AUC <- Wilcoxon(zk1, zk2)

z <- (AUC - muAUC)/sigmaAUC

p <- 2\*pnorm(-abs(z)) # p value for individual trial

if (p < alpha) reject[trial] = 1

}

CI <- c(0,0); width <- -qnorm(alpha/2)

ObsvdTypeIErrRate <- sum(reject)/length(reject)

CI[1] <- ObsvdTypeIErrRate -

width\*sqrt(ObsvdTypeIErrRate\*(1-ObsvdTypeIErrRate)/nTrials)

CI[2] <- ObsvdTypeIErrRate

+ width\*sqrt(ObsvdTypeIErrRate\*(1-ObsvdTypeIErrRate)/nTrials)

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

cat("ObsvdTypeIErrRate = ",

ObsvdTypeIErrRate, "\n95% confidence interval = ", CI, "\n")

exact <- binom.test(sum(reject),n = 2000,p = alpha)

cat("exact 95% CI = ", as.numeric(exact$conf.int), "\n")

Lines 15 – 24 conduct 2000 trials under the NH condition, i.e., the dataset was obtained, line 19, from the same distributions as the population. Line 20 calculates the AUC and lines 21-22 calculate the z-statistic and the p-value. If the p-value is smaller than the alpha of the test, currently set at 0.05, line 23 stores a 1 in a zero-initialized 2000 element array. Lines 26 – 36 calculate the observed rejection fraction and the corresponding exact 95% confidence interval. If the hypothesis testing procedure is valid, the observed interval should include 0.05.

### Online Appendix 8.B.2: Code output

> source(...)

alpha = 0.05

ObsvdTypeIErrRate = 0.049

95% confidence interval = 0.03953934 0.05846066

exact 95% CI = 0.03995676 0.05939265

The procedure passes the NH validation test. For the chosen value of alpha (0.05) the approximate and exact confidence intervals are similar.

# Online Appendix 8.C: Metz's ROC within an ROC

This relates to book Section 8.5.1. A listing of file mainRocWithinRoc.R follows:

### Online Appendix 7.C.1: Code Listing

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

library(ggplot2)

source("Wilcoxon.R")

seed <- 1;set.seed(seed)

muNH <- 1.5;muAH <- 2.1;sigma <- 1.3

K1 <- 50;K2 <- 52#;K1 <-K1\*2;K2 <- K2\*2

# cheat to find the population mean and std. dev.

AUC <- array(dim = 10000)

for (i in 1:length(AUC)) {

zk1 <- rnorm(K1);zk2 <- rnorm(K2, mean = muNH, sd = sigma)

AUC[i] <- Wilcoxon(zk1, zk2)

}

sigmaAUC <- sqrt(var(AUC));meanAUC <- mean(AUC)

T <- 2000

mu <- c(muNH,muAH)

alphaArr <- seq(0.05, 0.95, length.out = 10)

EmpAlpha <- array(dim = length(alphaArr))

EmpPower <- array(dim =length(alphaArr))

for (a in 1:length(alphaArr)) { # a is index into alpha array

alpha <- alphaArr[a]

reject <- array(0, dim = c(2, T))

for (h in 1:2) {

for (t in 1:length(reject[h,])) {

zk1 <- rnorm(K1)

zk2 <- rnorm(K2, mean = mu[h], sd = sigma)

AUC <- Wilcoxon(zk1, zk2)

obsvdZ <- (AUC - meanAUC)/sigmaAUC

p <- 2\*pnorm(-abs(obsvdZ)) # p value for individual a

if (p < alpha) reject[h,t] = 1

}

}

EmpAlpha[a] <- sum(reject[1,])/length(reject[1,])

EmpPower[a] <- sum(reject[2,])/length(reject[2,])

}

# plot the data

EmpAlpha <- c(0,EmpAlpha,1)

EmpPower <- c(0,EmpPower,1)

pointData <- data.frame(EmpAlpha = EmpAlpha, EmpPower = EmpPower)

zetas <- seq(-5, 5, by = 0.01)

muRoc <- 1.8 # found by trial and error

curveData <- data.frame(EmpAlpha = pnorm(-zetas),

EmpPower = pnorm(muRoc - zetas))

alphaPowerPlot <- ggplot(mapping = aes(x = EmpAlpha, y = EmpPower)) +

geom\_point(data = pointData, shape = 1, size = 3) +

geom\_line(data = curveData)

print(alphaPowerPlot)

The code is explained in the book. Sourcing it yields book Figure 8.1.

# Online Appendix 8.D: Factors affecting statistical power

This relates to book Section 8.5.2. A listing of file mainStatPower.R follows:

### Online Appendix 7.D.1: Code Listing

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

source("Wilcoxon.R");source("EffectSize.R")

seed <- 1;set.seed(seed)

mu <- 1.5;muAH <- 2.1;sigma <- 1.3

K1 <- 50;K2 <- 52#;K1 <- K1\*2;K2 <- K2\*2

# cheat to find the population mean and std. dev.

AUC <- array(dim = 10000)

for (i in 1:length(AUC)) {

zk1 <- rnorm(K1)

zk2 <- rnorm(K2, mean = mu, sd = sigma)

AUC[i] <- Wilcoxon(zk1, zk2)

}

sigmaAUC <- sqrt(var(AUC));muAUC <- mean(AUC)

T <- 2000

alpha <- 0.01 # size of test

reject = array(0, dim = T)

for (t in 1:length(reject)) {

zk1 <- rnorm(K1)

zk2 <- rnorm(K2, mean = muAH, sd = sigma)

AUC <- Wilcoxon(zk1, zk2)

obsvdZ <- (AUC - muAUC)/sigmaAUC

p <- 2\*pnorm(-abs(obsvdZ)) # p value for individual t

if (p < alpha) reject[t] = 1

}

ObsvdTypeIErrRate <- sum(reject)/length(reject)

CI <- c(0,0);width <- -qnorm(alpha/2)

CI[1] <- ObsvdTypeIErrRate -

width\*sqrt(ObsvdTypeIErrRate\*(1-ObsvdTypeIErrRate)/T)

CI[2] <- ObsvdTypeIErrRate +

width\*sqrt(ObsvdTypeIErrRate\*(1-ObsvdTypeIErrRate)/T)

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

cat("#non-diseased images = ", K1,

"\n#diseased images = ", K2, "\n")

cat("obsvdPower = ", ObsvdTypeIErrRate, "\n")

cat("95% confidence interval = ", CI, "\n")

cat("Effect Size = ", EffectSize(mu, sigma, muAH, sigma), "\n")

This code is similar to mainTypeIErrors.R, except that this time the datasets are simulated from distributions with a larger AUC (muAH > mu). Sourcing the code yields:

### Online Appendix 8.D.2: Code output

> source(...)

alpha = 0.05

#non-diseased images = 50

#diseased images = 52

obsvdPower = 0.509

95% confidence interval = 0.4870905 0.5309095

Effect Size = 0.08000617

One should experiment with the parameters of the simulation model (numbers of cases, effect size, alpha) to confirm the statements in the book about statistical power, book Section 8.5.2.