Online Appendix 4.A: Operating points from counts table

After opening the **software.Rproj** file in the **software** directory corresponding to this chapter, open the file **mainOpPtsFromCountsTable.R**, the listing for which follows:

Online Appendix 4.A.1: code listing

```
# mainOpPtsFromCountsTable.R
rm( list = ls())
source("plotROC.R");library(ggplot2)
options(digits = 4)
Ktr = array(dim = c(2,5))
Ktr[1,]  <- c(30,19,8,2,1) # Table 4.2.1
Ktr[2,]  <- c(5,6,5,12,22)

R <- length(Ktr[1,]) -1 # # number of op points; 4

FPF <- array(0,dim = R)
TPF <- array(0,dim = R)

for (r in (R+1):2) {
  FPF[(R+2)-r] <- sum(Ktr[1, r:(R+1)])/sum(Ktr[1,])
  TPF[(R+2)-r] <- sum(Ktr[2, r:(R+1)])/sum(Ktr[2,])
}

cat("FPF =", "\n")
cat(FPF, "\n")
cat("TPF = ", "\n")
cat(TPF, "\n")

mu <- qnorm(.5)+qnorm(.9);sigma <- 1
Az <- pnorm(mu/sqrt(2))
plotROC (mu, sigma, FPF, TPF)
cat("uppermost point based estimate of mu = ", mu, "\n")
cat("corresponding estimate of Az = ", Az, "\n")
cat("showing observed operating points and eq. var. fitted ROC curve", "\n")

mu <- 2.17;sigma <- 1.65
Az <- pnorm(mu/sqrt(1+sigma^2))
plotROC (mu, sigma, FPF, TPF)
cat("binormal estimate of Az = ", Az, "\n")
cat("showing observed operating points and uneq. var. fitted ROC curve", "\n")
```

Lines 5-7 define the appropriately named variable **RocCountsTable**, a 2 x 5 array, which contains the count data in Table 4.1. Line 9 defines **R**, the number of operating points, 4 in this case (note the minus one). When a particular value is used in several places it is better to define it as a variable, so if a change is made to the definition of the variable the values, wherever else the number is used, will be automatically updated. Line 11 - 12 initializes, to zeroes, two arrays **FPF** and **TPF** of length 4, each. Regarding line 14, **(R+1):2** is the

sequence of integers, starting from 5 and ending at 2, i.e., 5, 4, 3,2 (**R** knows to "step backwards"). The **for**-loop beginning on line 14 says, effectively, that the statements in the curly brackets are to be repeated for the following successive values of **r**: 5, 4, 3 and 2. Lines 15 - 16 implement the cumulating and copies the 4 pairs of values to the appropriate positions in the array variables **FPF** and **TPF**. Since this appears a little complicated, let us work through it and in doing so, illustrate a simple way of understanding code in **R** by literally running it line-by-line. Insert a break point at line 14 and click on **Source**. Click on **Next**; the cursor advances to line 15. Observe in the Environment window that **r** has the value 5. Now, on line 15, highlight only **RocCountsTable[1, r:(R+1)]** and click on **Run**. You should see the following:

Online Appendix 4.A.2: code snippet

```
Browse[2]> RocCountsTable[1, r:(R+1)]
[1] 1
Browse[2]>
```

This is because **r:(R+1)** is 5:5, which is 5 (in **R** the colon is the sequence operator; the more general form is the **seq()** function, which takes longer to write but gives you more control). So we are printing **RocCountsTable[1,5]**, which happens to be 1 (see line 6). The **sum()** function on a single variable argument with value 1 gives 1. On the other hand the denominator of this line gives 60 (the sum of the array defined on the right hand side of line 6); instead of error-prone typing, simply highlight and click on **Run**. You should see:

Online Appendix 4.A.3: code snippet

```
Browse[2]> sum(RocCountsTable[1,])
[1] 60
Browse[2]>
```

The ratio gives the x-coordinate of the lowest operating point (corresponding to the highest threshold), namely 0.01667, which is assigned to **FPF[1]**.

Online Appendix 4.A.4: code snippet

```
Browse[2]> sum(RocCountsTable[1, r:(R+1)])/sum(RocCountsTable[1,])
[1] 0.01667
Browse[2]>
```

Click on **Next**. The right hand side of line 15 gives 0.44, which is assigned to **TPF[1]**. Click on **Next** again to start executing the 2nd iteration of the **for**-loop. On line 14 highlight only **RocCountsTable[1, r:(R+1)]** and click on **Run**. You should see the following:

Online Appendix 4.A.5: code snippet

```
Browse[2]> RocCountsTable[1, r:(R+1)]
```

```
[1] 2 1
Browse[2]>
```

You can see it has picked up one more element from the array defined in line 6, so the sum in the numerator will yield 3, which divided by 60 gives the x-coordinate of the next-higher operating point, i.e., 0.05. Keep clicking on **Next** and use highlighting and **Run** to confirm that the code is correctly calculating the operating points listed in Table 4.1. Eventually the code will exit the for-loop.