Notes on the Bioconductor `ROC` library

April 25, 2006

1 Introduction

The `ROC` library is a collection of R classes and functions related to receiver operating characteristic (ROC) curves. These functions are targeted at the use of ROC analysis with DNA microarrays, as discussed in work of MS Pepe, G Anderson and colleagues at U Washington Seattle Biostatistics (unpublished report). Other open source software for ROC analysis in the S language has been distributed by Beth Atkinson of the Mayo Clinic.

2 Inventory of classes and functions

2.1 `rocc`: a representation of an ROC curve

ROC curves illustrate the performance of a classification procedure based on dichotomous interpretation of a continuous marker. Suppose the true state of an item is either '+' or '−'. The item's marker value is compared to a threshold, and the item is classified as positive or negative depending on whether the marker value is greater than or less than the threshold. For a fixed threshold t, the procedure has a sensitivity Pr(marker $\geq t$ |true class= +) and specificity Pr(marker $< t$ |true class= −). The ROC curve is the locus of values $(x, y) = (1 - \text{spec}, \text{sens})$.

`rocc` is an S4-style class, with slots

- `sens`, vector of sensitivity values
- `spec`, vector of specificity values
- `rule`, archival value of the rule used to classify items
- `cuts`, vector of thresholds used
- `markerlabel`, name of the marker
- `caselabel`, name of the state

Let's verify:

```
> library(ROC)
> print(getClass("rocc"))

Slots:

Name:         sens         spec         rule         cuts markerLabel   caseLabel
Class:     numeric      numeric     function      numeric    character   character
```

For creation of an ROC curve object (an instance of class `rocc`), the function `rocdemo.sca` is available. The name is chosen to indicate that this is a provisional definition based on a scalar marker.

`rocdemo.sca` is defined as follows:

```
> print(rocdemo.sca)

function (truth, data, rule, cutpts = NA, markerLabel = "unnamed marker",
    caseLabel = "unnamed diagnosis")
{
    if (!all(sort(unique(truth)) == c(0, 1)))
        stop("'truth' variable must take values 0 or 1")
    if (is.na(cutpts)) {
        udata <- unique(sort(data))
        delta <- min(diff(udata))/2
        cutpts <- c(udata - delta, udata[length(udata)] + delta)
    }
    np <- length(cutpts)
    sens <- rep(NA, np)
    spec <- rep(NA, np)
    for (i in 1:np) {
        pred <- rule(data, cutpts[i])
        sens[i] = mean(pred[truth == 1])
        spec[i] = mean(1 - pred[truth == 0])
    }
    new("rocc", spec = spec, sens = sens, rule = rule, cuts = cutpts,
        markerLabel = markerLabel, caseLabel = caseLabel)
}
<environment: namespace:ROC>
```

The argument `truth` is the vector of actual states of the objects being classified, and it must be a vector of binary indicators. The argument `data` is the vector of marker values. The argument `rule` is the classification rule. This must be a function of two arguments, and the following is an obvious approach:

```
> print(dxrule.sca)

function (x, thresh)
ifelse(x > thresh, 1, 0)
<environment: namespace:ROC>
```

```
> plot(roc1)
```

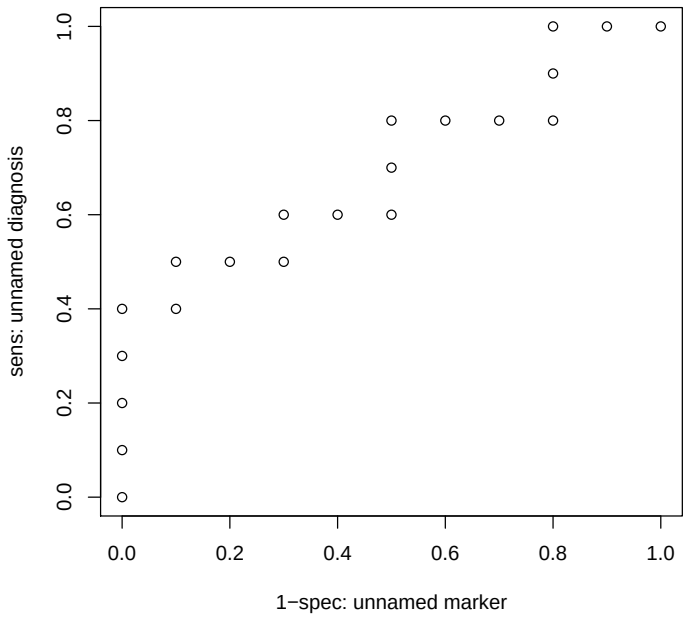

Figure 1: An example ROC curve with default plotting parameters.

To begin working with ROC curves in R, you can proceed as follows.

```
> set.seed(123)
> state <- c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1,
+     1, 1, 1)
> markers <- c(1, 2, 1, 1, 2, 3, 3, 4, 2, 1, 1, 3, 2, 3, 2, 4,
+     5, 2, 3, 4) + runif(20, -1, 1)
> roc1 <- rocdemo.sca(truth = state, data = markers, rule = dxrule.sca)
```

2.2 Functionals of the ROC curve

The area under the ROC curve is well known to be equivalent to the numerator of the Mann-Whitney U statistic comparing the marker distributions among positive and negative items. Even if the overall area is not very large, the existence of threshold values yielding sensitivities markedly greater than the false positive rate can be of interest. The sensitivity at a false positive rate t is

denoted ROC(t); the AUC $= \int_0^1 ROC(u)du$. Finally, the partial AUC to a false positive rate s is denoted pAUC(s) $= \int_0^s ROC(u)du$.

```
> auc <- AUC(roc1)
> print(auc)
```

```
[1] 0.7
```

```
> paucp4 <- pAUC(roc1, 0.4)
> print(paucp4)
```

```
[1] 0.2
```

```
> rocp3 <- ROC(roc1, 0.3)
> print(rocp3)
```

```
[1] 0.6
```

Note that the definition of the AUC uses a naive trapezoidal rule. This is faster than `integrate`. However, functions AUCi and pAUCi that use the more accurate integrate() function are available.

```
> print(trapezint)
```

```
function (x, y, a, b)
{
    if (length(x) != length(y))
        stop("length x must equal length y")
    y <- y[x >= a & x <= b]
    x <- x[x >= a & x <= b]
    if (length(unique(x)) < 2)
        return(NA)
    ya <- approx(x, y, a, ties = max, rule = 2)$y
    yb <- approx(x, y, b, ties = max, rule = 2)$y
    x <- c(a, x, b)
    y <- c(ya, y, yb)
    h <- diff(x)
    lx <- length(x)
    0.5 * sum(h * (y[-1] + y[-lx]))
}
<environment: namespace:ROC>
```

2.3 Tools for working with microarrays

See the Biobase package for a discussion of the `exprSet` class, which represents a collection of microarrays. Given a dichotomous element of the `phenodata` slot/data-frame of an `exprSet`, an ROC curve may be defined using the expression levels of any gene as the vector of marker values.

We confine activities to the first 50 genes in eset so that this vignette is computable in reasonable CPU time.

```
> library(Biobase)
> data(eset)
> myauc <- function(x) {
+     dx <- cov1 - 1
+     AUC(rocdemo.sca(truth = dx, data = x, rule = dxrule.sca))
+ }
> mypauc1 <- function(x) {
+     dx <- cov1 - 1
+     pAUC(rocdemo.sca(truth = dx, data = x, rule = dxrule.sca),
+         0.1)
+ }
> allAUC <- esApply(eset[1:50, ], 1, myauc)
> allpAUC1 <- esApply(eset[1:50, ], 1, mypauc1)
> print(geneNames(eset[1:50, ])[allAUC >= max(allAUC, na.rm = TRUE) &
+     !is.na(allAUC)])

[1] "AFFX-PheX-5_at"

> print(geneNames(eset[1:50, ])[allpAUC1 >= max(allpAUC1, na.rm = TRUE) &
+     !is.na(allpAUC1)])

[1] "AFFX-HUMGAPDH/M33197_3_at"
```

2.4 Inference on ROC-based rankings

Pepe et al indicate that rankings based on ROC functionals are of interest, and that uncertainty in rankings can be exposed by resampling tissues. Working with exprSets, this is easy to carry out. (We use a very small number of resamplings (5) to get through this in a reasonable execution time.)

```
> print("code commented out owing to odd CHECK error")

[1] "code commented out owing to odd CHECK error"

> print("will run in an interactive session")

[1] "will run in an interactive session"

> if (FALSE) {
+     nResamp <- 5
+     nTiss <- ncol(exprs(eset))
+     nGenes <- nrow(exprs(eset[1:50, ]))
+     out <- matrix(NA, nr = nGenes, nc = nResamp)
+     set.seed(123)
+     for (i in 1:nResamp) {
+         TissInds <- sample(1:nTiss, size = nTiss, replace = TRUE)
+         out[, i] <- esApply(eset[1:50, TissInds], 1, myauc)
+     }
+     rout <- apply(out, 2, rank)
+ }
```

Each row of the matrix `rout` is a sample from the bootstrap distribution of AUC ranks for the corresponding gene.