Calling individual modules of arrayQualityMetrics

Audrey Kauffmann

December 27, 2010

Contents

Introduction

In the vignette arrayQualityMetrics.pdf, you can learn how to use the arrayQualityMetrics function to create a full report. In this vignette, we show how to only run one or several of the metrics and how to add your own modules to customize the report.

1 Functions to format the data for aqm functions usage

There are two functions to prepare the data. They compute intermediate results that are then processed by the `aqm.xxx` functions. They are greedy for memory and slow, but they only need to run once, and this modular design allows us more flexibility and extensibility with the report generation.

- `aqm.prepdata`: generates a *aqmobj.prepdata* object which can be the input of all the `aqm.xxx` functions.
- `aqm.prepaffy`: generates a *aqmobj.prepaffy* object which is the input for some of the Affymetrix specific functions.

2 Metrics available in the report

2.1 Functions with outlier detection

The functions that compute quality metrics are the following:

- `aqm.maplot`: computes the M versus A plot and generates an object *aqmobj.ma*.
- `aqm.spatial`: computes the spatial distribution of the ranks for the foreground intensities and generates an object *aqmobj.spatial*.
- `aqm.boxplot`: computes the boxplots of the intensities and generates an object *aqmobj.box*.
- `aqm.heatmap`: computes the heatmap of the between-array distances and generates an object *aqmobj.heatmap*.

The output objects contain the plot, the legend, the title, the name of the section it belongs to in the report, the scores per array that are used to perform the outlier detection and the array names that are detected as outliers.

2.2 Functions without outlier detection

The following functions also compute quality metrics:

- `aqm.spatialbg`: represents the spatial distribution of the rank background intensities and generates an object *aqmobj.spatialbg*.
- `aqm.density`: performs the density plots of the intensities and generates an object *aqmobj.dens*.

- `aqm.meansd`: compares the mean of the intensities to their standard deviation and generates an object *aqmobj.msd.*

- `aqm.probesmap`: draws density plots of two classes of probes as annotated in the "has-Target" column of the *featureData* and generates an object *aqmobj.probesmap.*

The output objects contain the plot, the legend, the title and the name of the section it belongs to in the report. No outlier detection is computed for those metrics.

2.3 Functions for Affymetrix data

The functions available for Affymetrix datasets are:

- `aqm.rnadeg`: is used to represent the RNA degradation plot and creates the object *aqmobj.rnadeg.*

- `aqm.rle`: performs Relative Log Expression boxplots and formats the output with outliers in the *aqmobj.rle* object.

- `aqm.nuse`: computes Normalised Unscaled Standard Error boxplots and outlier detection on it and formats the output as an object *aqmobj.nuse.*

- `aqm.qcstats`: draws the QCStats plot from the *simpleaffy* package and creates an object of class *aqmobj.qcs.*

- `aqm.pmmm`: draws density plots of the perfect match probes and the mismatch probes and generates an object*aqmobj.pmmm.*

3 Rendering of the output in a report

The functions available to render the output:

- `aqm.plot`: allows to plot in a R device any object of class *aqmobj.xxx* which contains a slot names `"plot"`.

- `aqm.writereport`: writes a html report containing the input metrics.

4 Example of a customized report

We will create a report with only 2 quality metrics, the boxplots and the density plots. First we need to load the needed packages and the data.

```
> library("arrayQualityMetrics")
> library("CCl4")
> library("vsn")
> data(CCl4)
> nCCl4 = justvsn(CCl4, subsample=2200)
```

```
vsn2: 44290 x 36 matrix (1 stratum). Please use 'meanSdPlot' to verify the fit.
```

We need to create an object *aqmobj.prepdata* to be able to call the quality metrics functions.

```
> dataprep = aqm.prepdata(expressionset = nCCl4, do.logtransform = FALSE)
```

We compute the boxplot and density plots metrics and then create a list containing both of them.

```
> bo = aqm.boxplot(nCCl4, dataprep, intgroup = c())
> de = aqm.density(nCCl4, dataprep, intgroup = c(), outliers = c())
> qm = list("Boxplot" = bo, "Density" = de)
```

We now can write a small report containing those two metrics.

```
> aqm.writereport(name = "Customized example", expressionset = nCCl4, obj = qm)
```

5 How to add your own metrics

Here is an example of a function, studying the GC content effects, that can be added to the report.

We are writing the function `aqm.GC` to compute boxplots grouped by GC content of the probes.

5.1 Creating the aqm function and aqmobj class

The first step is to create a class for the output object.

```
>  setClass("aqmobj.GC", representation(plot="list",
+                                       section="character",
+                                       title="character",
+                                       legend="character",
+                                       shape="list"))

[1] "aqmobj.GC"
```

Then, we can write the function. For the example, we will only make it work for *NChannelSet*. 'GC' refers to the GC content whereas 'gc' stands for 'green channel' in the data set (with 'rc' being 'red channel' and 'dat' the log(ratio)).

```
> setGeneric("aqm.GC",
+            function(expressionset, dataprep, ...)
+            standardGeneric("aqm.GC"))

[1] "aqm.GC"

> setMethod("aqm.GC",signature(expressionset = "NChannelSet"),
+           function(expressionset, dataprep, ...) {
+   fac = round(featureData(expressionset)$GC,-1)
+   df = data.frame(rep(fac, ncol(dataprep$dat)), as.numeric(dataprep$rc),
+     as.numeric(dataprep$gc), as.numeric(dataprep$dat))
+   colnames(df) = c("GC","rc","gc","dat")
+
+   box1 = bwplot(rc ~ GC, data = df, pch = "|", do.out = FALSE,
+     box.ratio = 2, xlab = "Red Intensity", ylab = "GC content",
+     horizontal = FALSE, fill="red")
+   box2 = bwplot(gc ~ GC, data = df, pch = "|", do.out = FALSE,
+     box.ratio = 2, xlab = "Green Intensity", ylab = "",
+     horizontal = FALSE, fill="green")
+   box3 = bwplot(dat ~ GC, data = df, pch = "|", do.out = FALSE,
+     box.ratio = 2, xlab = "Log(ratio)", ylab = "",
+     horizontal = FALSE, fill="blue")
+
+   plotGC = list("box1" =  box1, "box2" = box2, "box3" = box3)
+
+   title = "GC content effect"
+
+   section = "Probe stratification"
+
+   legend = "<b>Figure <!-- FIG --></b> shows the boxplots of the log<sub>2</sub>
+ intensities grouped by the percentage of cytosines (C) and guanines (G) among the
+ nucleotides in each probe. Cytosine and guanine are able to form three hydrogen bonds,
+ while adenine (A) and thymine (T) only form two, hence oligonucleotides with a higher
+ proportion of C and G can form more stable hybridization bindings.
+ This tends to result in higher intensities measured on the array, regardless of the
+ abundance of target molecules."
+
+   out = list("plot" = plotGC, "section" = section, "title" = title,
+              "legend" = legend, "shape" = list(h=9,w=4))
+   class(out) = "aqmobj.GC"
+   return(out)
+ })

[1] "aqm.GC"
```

5.2 Creating a method for aqm.plot

Once the `aqm.GC` is written, we need to write a `aqm.plot` method for it, to be able to draw the plot with the desired settings.

```
> setMethod("aqm.plot",signature(obj = "aqmobj.GC"), function(obj){
+   print(obj$plot$box1, split = c(1,1,3,1), newpage = FALSE)
+   print(obj$plot$box2, split = c(2,1,3,1), newpage = FALSE)
+   print(obj$plot$box3, split = c(3,1,3,1), newpage = FALSE)
+ })

[1] "aqm.plot"
```

5.3 Example of application

We can call the function on an example dataset. We first create a column "GC" in the *featureData* of the object `nCCl4`.

```
> datapath = system.file("extdata", package="CCl4")
> seq = read.AnnotatedDataFrame("013162_D_SequenceList_20060815.txt", path=datapath)
> if(any(duplicated(featureNames(seq))))
+   cat("IDs of the sequence file are not unique \n")
> library("Biostrings")
> bc = alphabetFrequency(DNAStringSet(seq$Sequence), baseOnly=TRUE)
> GC = ((bc[,"C"]+bc[,"G"])/rowSums(bc))*100
> mt = match(featureNames(seq), fData(nCCl4)$ID)
> stopifnot(!any(is.na(mt)))
> fData(nCCl4)$GC = NA_real_
> fData(nCCl4)$GC[mt] = GC
```

Now, we can call the `aqm.GC` function and create a small html report with its output.

```
> library("lattice")
> dataprep = aqm.prepdata(nCCl4, FALSE)
> testgc = aqm.GC(nCCl4,dataprep)
> aqm.plot(testgc)
> aqm.writereport("GC Study", nCCl4, testgc)
```

Session Info

```
> toLatex(sessionInfo())
```

- R version 2.12.1 (2010-12-16), `x86_64-apple-darwin10.5.0`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: Biobase 2.10.0, Biostrings 2.18.2, CCl4 1.0.9, IRanges 1.8.7, affy 1.28.0, affyPLM 1.26.0, arrayQualityMetrics 3.2.3, fortunes 1.4-0, gcrma 2.22.0, lattice 0.19-13, limma 3.6.9, preprocessCore 1.12.0, vsn 3.18.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.12.0, DBI 0.2-5, RColorBrewer 1.0-2, RSQLite 0.9-4, SVGAnnotation 0.7-2, XML 3.2-0, affyio 1.18.0, annotate 1.28.0, beadarray 2.0.2, genefilter 1.32.0, grid 2.12.1, hwriter 1.3, latticeExtra 0.6-14, marray 1.28.0, simpleaffy 2.26.1, splines 2.12.1, stats4 2.12.1, survival 2.36-2, tools 2.12.1, xtable 1.5-6