Inside tami

Guillermo Ayala

5/11/23

Table of contents

Objective

- A package for students with a low level in R programming.
- A focus in differential expression using RNA-seq and microarrays.
- Use many Bioconductor packages.
- Main interest in statistical analysis.

Loading tami

```
pacman::p_load(Biobase,tami)
```

S4 classes

- Different steps of the statistical analysis represented as classes.
- S4 classes because Bioconductor use this system.

ExpressionData

- Basic information:
    1. Expression matrix.
    2. Phenotipic variable or covariable: two-level factor.
    3. Type of data: not used.

```
#' @importFrom  methods new
NULL

#' @title ExpressionData
#' S4 class
#' @description
#' S4 class with the basic information
#' @export ExpressionData
#' @exportClass ExpressionData
ExpressionData = setClass("ExpressionData",slots =
                            list(exprm  = "matrix",
                                 groups = "factor",
                                 type   = "character"))
```

Making an ExpressionData

```
exprm0 = matrix(rnorm(100),ncol=5)
rownames(exprm0) = letters[1:20]
groups0 = factor(c(1,2,2,1,1),levels = 1:2,labels=c("A","B"))
x  = new("ExpressionData",exprm = exprm0,groups=groups0)
```

```
data(gse21942,package="tamidata")
x = ExpressionData(exprm = exprs(gse21942),
                       groups = pData(gse21942)[,"FactorValue..DISEASE.STATE."],
                       type="microarray")
```

Generic function

```
#' Generic function for \code{exprm}
#' @description
#' Accesor for \code{exprm}
#' @param object \code{ExpressionData}
#' @export
#'
setGeneric("exprm", function(object) standardGeneric("exprm"))

#' Generic function for \code{exprm<-}
#' @description
#' Generic function for \code{exprm<-}
#' @param object \code{exprm} slot of \code{ExpressionData}
#' @param value for \code{exprm} slot of \code{ExpressionData}
#' @export
#'
setGeneric("exprm<-",
           function(object, value) standardGeneric("exprm<-"))
```

Method -> generic function -> class

```
#' Accesor for \code{exprm}
#' @description
#' Accesor for \code{exprm}
#' @param object \code{ExpressionData}
#' @export
#'
setMethod("exprm", signature = "ExpressionData",
          function(object) object@exprm)


#' Accesor for \code{exprm<-}
```

```
#' @description
#' Accesor for \code{exprm<-}
#' @param object exprm slot of ExpressionData
#' @param value for exprm slot of ExpressionData
#' @export
#'
setMethod("exprm<-", "ExpressionData",
          function(object, value) {
            object@exprm <- value
            if (methods::validObject(object))
              return(object)
          })
```

Using accesors

```
exprm(x)
```

```
groups(x)
```

```
 [1] multiple sclerosis multiple sclerosis multiple sclerosis multiple sclerosis
 [5] multiple sclerosis multiple sclerosis multiple sclerosis multiple sclerosis
 [9] multiple sclerosis multiple sclerosis multiple sclerosis multiple sclerosis
[13] multiple sclerosis multiple sclerosis healthy            healthy
[17] healthy            healthy            healthy            healthy
[21] healthy            healthy            healthy            healthy
[25] healthy            healthy            healthy            healthy
[29] healthy
Levels: healthy multiple sclerosis
```

DifferentialExpressionInput

```
#' DifferentialExpressionInput
#' S4 class for  differential expression analysis
#' @description
#' S4 class with the basic information to perform an differential expression
#' analysis
#' @slot correction The correction method for multiple comparisons (as used
#' by \link[stats]{p.adjust}).
```

```
#' @slot test Statistical method used for marginal differential expression
#' @slot association Statistic measuring expression-phenotype association
#' @slot nulldistr Null distribution tested c("exact","randomization",
#' "bootstrap")
#' @export DifferentialExpressionInput
#' @exportClass DifferentialExpressionInput
#'

DifferentialExpressionInput = setClass("DifferentialExpressionInput",
         slots = list(correction = "character",
                      test = "character",
                      association = "character",
                      nulldistr = "character"),
         contains = "ExpressionData")
```

Making ...

```
x_dei = DifferentialExpressionInput(exprm =exprm(x),
                 groups =  groups(x),association ="pvalue",
                 correction = "BH")
```

```
x_dei = new("DifferentialExpressionInput",exprm =exprm(x),
                 groups =  groups(x),association ="pvalue",
                 correction = "BH")
```

DifferentialExpressionOutput

```
#' \code{DifferentialExpressionOutput} S4 class
#' @description
#' S4 class with the basic statistical association of a statistical analysis
#' of differential expression analysis
#' @slot GeneData Data frame containing different gene identifiers
#' named "ENTREZID", "ENSEMBL", "PROBEID"
#' @slot GeneStat Data frame with columns \code{statistic}, \code{rawp},
#' \code{adjp}, \code{qval}
#' corresponding to the marginal differential expression analysis: the original
#' statistic, the raw p-value, the adjusted p-value and the q-value
#' @slot foutput File to save results
```

```
#' @slot fdr False discovery rate
#' @slot userGeneSet  The vector indices (rows of expression matrix)
#' indicating the genes in which the user is interested
#' @export DifferentialExpressionOutput
#' @exportClass DifferentialExpressionOutput
#'
DifferentialExpressionOutput = setClass("DifferentialExpressionOutput",
         slots = list(GeneData    =  "data.frame",
                      GeneStat    =  "data.frame",
                      foutput     = "character",
                      fdr         = "numeric",
                      userGeneSet = "vector"),
         contains = "DifferentialExpressionInput")
```

Making ...

```
#' Differential expression for microarrays and RNA-seq data
#' @description
#' Differential expression for microarrays and RNA-seq data
#' @param x \code{ExpressionSet} or \code{RangedSummarizedExperiment}
#' @param y Name of a two-level factor
#' @param test Testing procedure for marginal differential expression.
#' It should return a \code{data.frame} with two columns named
#' \code{statistic} and \code{rawp}. Some possible \code{test}'s are
#' \link{rowt},\link{rowtmod},\link{edgercommon},\link{edgertagwise}
#' @param correction Type of correction for multiple testing
#' used by  \code{stats::p.adjust}
#' @param fdr False discovery rate
#' @param foutput File for html report
#' (\code{stats::p.adjust})
#' @export
dema = function(x,y,test,
                correction=c("BH","bonferroni","holm","hochberg", "hommel",
                             "BH", "BY","none"),
                fdr= 0.01,
                foutput = "output"){
  xed = convert(x,y)
  if(type(xed) == "ExpressionSet"){
    if(ncol(Biobase::fData(x))==0) Biobase::fData(x) = Biobase::featureNames(x)
    xa = Biobase::fData(x)
```

```
  }
  if(type(xed) == "RangedSummarizedExperiment"){
    if(ncol(SummarizedExperiment::rowData(x))==0)
      SummarizedExperiment::rowData(x) = rownames(x)
    xa = data.frame(SummarizedExperiment::rowData(x))
  }
  xt = rowtest(x=exprm(xed),y=groups(xed),test,correction)
  result = DifferentialExpressionOutput(GeneData = xa,GeneStat = xt,
                                        foutput=foutput,fdr=fdr,
                                        exprm=exprm(xed),groups=groups(xed))
  result
}
```

```
x1 = dema(x=gse21942,y="FactorValue..DISEASE.STATE.",correction = "BH",
          test = rowt,foutput = "gse21942",fdr = .01)
```

```
Warning: replacing previous import 'utils::findMatches' by
'S4Vectors::findMatches' when loading 'AnnotationDbi'
```

GSAInput

```
#' S4 class with the input to perform gene set analysis
#' @description
#' S4 class with the data to perform a gene set analysis
#' @slot  GeneSetList \code{list} containing the gene sets. Each gene set is an
#' element of the \code{list} and the \code{name} of this element is the name
#' of the
#' group with annotation data for the gene sets for instance, GO, KEGG, ...)
#' @slot EnrichmentMethod Enrichment method
#' @slot fdr False discovery rate
#' @details
#' The different \code{method}s implemented are
#' 1. \code{ora} Over-representation analysis with \code{SignificantGeneSet}
#' 2. Significant gene set is defined as set0 = which(GeneStat[,"qval"] < fdr)
#' and an over-representation analysis is performed.
#' 3. An over-representation analysis using as significant gene set
#' \code{SignificantGeneSet}
#' @export
#'
```

```
setClass("GSAInput",
         slots = list(GeneSetList        = "list",
                      EnrichmentMethod   = "character",
                      fdr                = "numeric"),
         contains = "DifferentialExpressionOutput")
```

GSAOutput

```
#' S4 class for output of gene set analysis
#' @description
#' S4 class to generate outputs of gene set analysis
#'
#' @slot  GeneSetData \code{data.frame} containing the gene set data (identifiers
#' from GO, KEGG,...)
#' @slot GeneSetStat \code{data.frame} giving the different associations from
#' expression profile and phenotipic variable: \code{statistic}, \code{rawp}, \code{adjp}
#' and \code{qval}.
#' @slot correction Method for correction in multiple comparisons
#' @slot foutput File where to save the results (.html)
#' @export
#'
setClass("GSAOutput",
         slots = list(GeneSetData        = "data.frame",
                      GeneSetStat        = "data.frame",
                      correction         = "character",
                      foutput            = "character"),
         contains = "GSAInput")
```

```
#' @title ora
#' Over-representation analysis
#' @description
#' Given a gene set \code{set0} and a list with a gene set collection,
#' an over-representation analysis is performed.
#' Each set corresponds with a  row subset.
#' @param set0 Significant genes
#' @param gsc List of gene set collection
#' @return A \code{data.frame} with the p-values, \code{rawp},
#' the observed odds ratio, \code{oddsratio}, and the confidence interval
#' for the odds ratio, [\code{oddsratio_lower},\code{oddsratio_upper}]
```

```
#' @export
ora = function(set0,gsc){
  N = length(unique(unlist(gsc)))
  np1 = sapply(gsc,length)
  np2 = N - np1
  n1p = length(set0)
  n2p = N - n1p
  n11 = sapply(gsc,function(set1,set0)sum(is.element(set1,set0)),set0)
  n21 = np1 - n11
  n12 = n1p - n11
  n22 = N - n11 - n12 - n21
  countmatrix = cbind(n11,n21,n12,n22)
  temp = t(pbapply::pbapply(countmatrix,1,fisherRow,alternative="greater"))
  result = data.frame(rawp = temp[,1],OR = temp[,2],
                      OR_low = temp[,3],OR_up  = temp[,4])
  result
}

#' Over-representation analysis
#' @description
#' Over-representation analysis
#' @param x \code{DifferentialExpressionOutput} (if NULL the significant gene
#' set is \code{SigGeneSet})
#' @param minsize Minimum size of the gene set
#' @param maxsize Maximum size of the gene set
#' @param correction Correction method for multiple comparisons
#' @param fdr False discovery rate
#' @param GeneSetList \code{list} of gene sets
#' @param SigGeneSet Significant gene set
#' @param foutput Output file for html report
#' @param id Type of identifier ("ENTREZID", "ENSEMBL") used in
#' \code{GeneSetList} and \code{SigGeneSet}
#' @export
#'
overRepresentation = function(x,minsize=5,maxsize=100,
                              correction = "BH",fdr = 0.01,
                              GeneSetList = NULL,
                              SigGeneSet = NULL,
                              foutput="GSAout",id ="ENTREZID"){
  ## Is x a DifferentialExpressionOutput object?
  if(class(x) != "DifferentialExpressionOutput")
```

```
    return("x must be a DifferentialExpressionOutput")
  if(is.null(GeneSetList))
    return("A list of gene sets is required")
  correction = match.arg(correction)
  ngs = sapply(GeneSetList,length)
  sel0 = which(ngs >= minsize & ngs <= maxsize)
  if(length(sel0) == 0) return("There is no gene set in [minsize,maxsize]")
  GeneSetList0 = GeneSetList[sel0]
  if(is.null(SigGeneSet))
    SigGeneSet = GeneData(x)[GeneStat(x)[,"adjp"] < fdr(x),id]
  ora0 = ora(SigGeneSet,GeneSetList0)
  temp1 = rawp2adjpq(ora0[,"rawp"],correction)

  new("GSAOutput",
      GeneSetData=data.frame(DB=names(GeneSetList)[sel0]),
      GeneSetStat=data.frame(ora0,temp1),correction = correction,
      foutput =foutput)
}
```

tidy, glimpse, augment

```
#' Generic function \code{glimpse}
#' @description
#' Glimpse statistical output mainly for html report generation
#' @param  object To be glimpse
#' @export
setGeneric("glimpse",def=function(object){standardGeneric("glimpse")})


#' Generic function \code{tidy}
#' @description
#' Tidy statistical output mainly for \code{data.frame} generation
#' @param  object To be tidy
#' @export
#'
setGeneric("tidy",def=function(object){standardGeneric("tidy")})

#' Generic function \code{augment}
#' @description
#' Generic function for an  statistical output augmented
```

```
#' @param  object  To be augmented
#'
#' @export
#'
setGeneric("augment",def=function(object){standardGeneric("augment")})
```

tidy, glimpse, augment for DifferentialExpressionOutput

```
#' @title tidy
#' Method for \code{DifferentialExpressionOutput}
#' @description
#' It generates a tidy report of the differential expression analysis
#' from a (S4 object) \code{DifferentialExpressionOutput}.
#' @param object  \code{DifferentialExpressionOutput}
#'
#' @export
#'
setMethod("tidy",
          signature = "DifferentialExpressionOutput",
          definition = function(object){
            sig = which(object@GeneStat[,"adjp"]< object@fdr)
            data.frame(object@GeneData[sig,],object@GeneStat[sig,])
          })


#' @title glimpse
#' Method for \code{DifferentialExpressionOutput}
#' @description
#' It generates a report in a html file with the results of
#' the differential expression analysis
#' from a (S4 object) \code{DifferentialExpressionOutput}.
#' @param object  \code{DifferentialExpressionOutput}
#'
#' @export
#'
setMethod("glimpse",
          signature = "DifferentialExpressionOutput",
          definition = function(object){
            if(is.element("ENTREZID",names(object@GeneData)))
              object@GeneData[,"ENTREZID"] =
```

```
            entrezid2url(object@GeneData[,"ENTREZID"])
          if(is.element("ENSEMBL",names(object@GeneData)))
            object@GeneData[,"ENSEMBL"] =
              ensembl2url(object@GeneData[,"ENSEMBL"])
          if(is.element("GO",names(object@GeneData)))
            object@GeneData[,"GO"] = go2url(object@GeneData[,"GO"])
          sig = which(object@GeneStat[,"adjp"]< object@fdr)

          df0 = data.frame(object@GeneData[sig,],object@GeneStat[sig,])
          htmlRep = ReportingTools::HTMLReport(
            shortName =object@foutput,title = object@foutput,
            reportDirectory = "./reports")
          ReportingTools::publish(df0,htmlRep)
          ReportingTools::finish(htmlRep)
        })



#' \code{augment} method for \code{DifferentialExpressionOutput}
#' @description
#' It generates a report in a html file with the results of
#' the differential expression analysis
#' from a (S4 object) \code{DifferentialExpressionOutput}.
#' Links to plots with a graphical display of the expression profile
#' is provided.
#' @param object  \code{DifferentialExpressionOutput}
#' @export
#'
setMethod("augment",
          signature = "DifferentialExpressionOutput",
          definition = function(object){
            if(!dir.exists("./reports/"))  dir.create("./reports/")
            if(!dir.exists("./reports/figures/"))
              dir.create("./reports/figures/")
            if(is.element("ENTREZID",names(object@GeneData)))
              object@GeneData[,"ENTREZID"] =
                entrezid2url(object@GeneData[,"ENTREZID"])
            if(is.element("ENSEMBL",names(object@GeneData)))
              object@GeneData[,"ENSEMBL"] =
                ensembl2url(object@GeneData[,"ENSEMBL"])
            #           if(is.element("GO",names(object@GeneData)))
```

```
#             object@GeneData[,"GO"] = tami::go2url(object@GeneData[,"GO"])

if(is.logical(object@userGeneSet))
  object@userGeneSet =
    sort(object@GeneStat[,"adjp"],index.return=TRUE)$ix[1:100]
rowPlot = function(i){
  df = data.frame(condition=object@groups,
                  expression=object@exprm[i,])
  p = ggplot2::ggplot(df, ggplot2::aes(x=df$condition,
                                       y=df$expression,
                                       color=df$condition))+
    ggplot2::geom_jitter(position=ggplot2::position_jitter(0.2))+
    ggplot2::stat_summary(fun.data="mean_sdl",geom="pointrange")+
    ggplot2::coord_flip()
  return(ggplot2::ggsave(filename = paste0(object@foutput,i,".png"),
                         plot=p,path = "./reports/figures/"))}
devnull = sapply(object@userGeneSet,rowPlot)
profile = vector("character",nrow(object@GeneData))
profile[object@userGeneSet] =
  paste0(getwd(),"/reports/figures/",object@foutput,
         object@userGeneSet,".png")
profile[object@userGeneSet] =
  paste0("<a href='file://",profile[object@userGeneSet],
         "'>","profile","</a>")
profile[-object@userGeneSet] = NA
sig = which(object@GeneStat[,"adjp"]<object@fdr)
sig = union(sig,object@userGeneSet)

df0 = data.frame(object@GeneData[sig,],object@GeneStat[sig,],
                 Profile = profile[sig])
htmlRep = ReportingTools::HTMLReport(
  shortName =object@foutput,title = object@foutput,
  reportDirectory = "./reports")
ReportingTools::publish(df0,htmlRep)
ReportingTools::finish(htmlRep)})
```

Using tidy, glimpse, augment

A tidy data.frame can be obtained using

```
x1_df = tidy(x1)
```

The same report in a html file is obtained with

```
glimpse(x1)
```

```
Warning: replacing previous import 'utils::findMatches' by
'S4Vectors::findMatches' when loading 'AnnotationForge'
```

```
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```

```
[1] "./reports/gse21942.html"
```

The html file just generated can be open with

```
browseURL(glimpse(x1))
```

or using the file manager.

setValidity

```
setValidity("ExpressionData", function(object) {
  msg = NULL
  valid = TRUE
  if(ncol(exprm(object)) != length(groups(object))){
    valid = TRUE
    msg <- c(msg,
             "The number of columns of exprm is not equal to the length of
             groups")
  }
  if (valid) TRUE else msg
  })
```

show, plot ...

```
#' @importMethodsFrom methods show

#' @title plot
#' Method for \code{ExpressionData}
#' @description
#' \code{plot} method for \code{ExpressionData}
#' @param x \code{ExpressionData}
#' @param y Type of plot:
#' \describe{
#' \item{1}{The means of log2-expression at x axis and
#' the estimated standard error of the difference of means at y axis}
#' \item{2}{The medians at x axis and the interquantile
#' ranges at y axis.}
#' \item{3}{The two first principal components with different colours
#' for the two groups. The package \code{ggfortify} is required.}
#' }
#' @param ... Additional parameters
#' @export
#'
setMethod(f = "plot",
          signature = c(x="ExpressionData",y="numeric"),
          definition = function(x,y,...){
            if(y == 1){
              mu0 = rowMeans(x@exprm)
              tt = genefilter::rowttests(x@exprm,x@groups)
              sd0 = tt[,"dm"]/tt[,"statistic"]
              df = data.frame(mu0,sd0)
              df = df[sort(mu0,index.return = TRUE)$ix,]
              return(ggplot2::ggplot(df,ggplot2::aes(x=mu0,y=sd0)) +
                       ggplot2::geom_smooth() +
                       ggplot2::xlab("Mean") +
                       ggplot2::ylab("Standard deviation"))
            }
            if(y == 2){
              median0 = matrixStats::rowMedians(x@exprm)
              iqr0 = matrixStats::rowIQRs(x@exprm)
              df = data.frame(median0,iqr0)
              df = df[sort(median0,index.return = TRUE)$ix,]
              return(ggplot2::ggplot(df,ggplot2::aes(x=median0,y=iqr0)) +
```

```
                ggplot2::geom_smooth() +
                ggplot2::xlab("Median") +
                ggplot2::ylab("Interquantile Range"))
    }
    ## Plot of the two first principal components
    if(y == 3){
      xt = t(x@exprm)
      df = data.frame(xt,groups = x@groups)
      ggplot2::autoplot(stats::prcomp(xt),data=df,colour="groups")
    }
    ## Tukey mean-difference for means per condition
    if(y == 4){
      mdt = vector("list",nlevels(x@groups))
      for(i in seq_along(levels(x@groups)))
        mdt[[i]] = rowMeans(x@exprm[,x@groups == levels(x@groups)[i]])

      mdt_mean = (mdt[[1]] + mdt[[2]])/2
      mdt_diff = mdt[[1]] - mdt[[2]]
      df = data.frame(mean = mdt_mean,difference = mdt_diff)
      df = df[sort(df$mean,index.return = TRUE)$ix,]
      ggplot2::ggplot(df,ggplot2::aes(x=df$mean,y=df$difference)) +
        ggplot2::geom_point() + ggplot2::ggtitle("Original scale")
      }
})
```