R for Bioinformatics

Introduction, Programming, Data Analysis and Visualization
Biological Data Analysis and Visualization

Gang Chen
chengang@bgitechsolutions.com

November 30, 2013

Outline

1. Bioconductor
2. S4 in Bioconductor
3. Genetic Variants Annotation in Bioconductor
4. Biological Data Visualization: ggbio

Next

1. Bioconductor
   - Overview
   - Installation
   - References
2. S4 in Bioconductor
3. Genetic Variants Annotation in Bioconductor
4. Biological Data Visualization: ggbio

What is Bioconductor

Bioconductor

- Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data.
- Bioconductor uses the R statistical programming language, and is open source and open development.
- It has two releases each year, 749 software packages, and an active user community.

Goals

- To provide widespread access to a broad range of powerful statistical and graphical methods for the analysis of genomic data. To facilitate the inclusion of biological metadata in the analysis of genomic data, e.g. literature data from PubMed, annotation data from Entrez genes.
- To provide a common software platform that enables the rapid development and deployment of extensible, scalable, and interoperable software.
- To further scientific understanding by producing high-quality documentation and reproducible research.
- To train researchers on computational and statistical methods for the analysis of genomic data.

Features

- The R Project for Statistical Computing.
- Documentation and reproducible research.
- Statistical and graphical methods.
- Annotation.
- Bioconductor short courses.
- Open source.
- Open development.

Core Team

- Vince Carey, Brigham & Women's, Harvard Medical School, USA.
- Marc Carlson Fred Hutchinson Cancer Research Center, USA.
- Sean Davis, USA, National Cancer Institute.
- Robert Gentleman, Genentech Research and Early Development, USA.
- Wolfgang Huber European Molecular Biology Laboratory, Heidelberg, Germany.
- Rafael Irizarry Department of Biostatistics, Johns Hopkins University
- Michael Lawrence, Genentech Research and Early Development, USA.
- James MacDonald, University of Michigan, USA.
- Martin Morgan, Fred Hutchinson Cancer Research Center, USA.
- Valerie Obenchain, Fred Hutchinson Cancer Research Center, USA.
- Herve Pages, Fred Hutchinson Cancer Research Center, USA.
- Paul Shannon, Fred Hutchinson Cancer Research Center, USA.
- Dan Tenenbaum, Fred Hutchinson Cancer Research Center, USA.

Installation

```
source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite(pkgname)
biocLite("BiocUpgrade")
```

References

Installation: http://bioconductor.org/install/

Help: http://bioconductor.org/help/

Package List: http://bioconductor.org/packages/release/BiocViews.html

Workflows: http://bioconductor.org/help/workflows/

Next

1 Bioconductor

2 S4 in Bioconductor
- Use of S4 in Bioconductor
- S4 Basics
- Case Study
- References

3 Genetic Variants Annotation in Bioconductor

4 Biological Data Visualization: ggbio

Statistics on packages in BioC 2.6

- 197 of 389 (51%) BioC packages define S4 classes
- 97 use inheritance
  - traditional parents: ExpressionSet and eSet from Biobase
  - newer parent: Sequence from IRanges
  - common parent: list, an S3 type
- 30 define virtual classes
- 14 use multiple inheritance

Important differences with Java and C++

- Classes don't own methods.
- Design goal: Be ``vectorized''.
  - Class slots ideally vectors and matrices.
  - Methods avoid unnecessary looping.

Class Creation

setClass function arguments

Class	the class name.
representation	the slot definitions.
contains	the parent classes.
prototype	the slot values for a default instance.
validity	a function that checks instance validity.

Object Creation

new constructor arguments

Class the class name.

... typically, slot values.

Class definition

```
setClass("Gene", # Class name
 representation =
  representation(name="character"), # slot definition
 prototype = prototype(name="BRAC1")) # default instance
```

Default Instantiation

```
new("Gene")

## An object of class "Gene"
## Slot "name":
## [1] "BRAC1"
```

Customized Instantiation

```
new("Gene", name = "TP53")

## An object of class "Gene"
## Slot "name":
## [1] "TP53"

new("Gene", name = "")

## An object of class "Gene"
## Slot "name":
## [1] ""

new("Gene", name = c("TP53", "EGFR"))

## An object of class "Gene"
## Slot "name":
## [1] "TP53" "EGFR"
```

Validaty Method

```
 setValidity("Gene", function(obj){
  if(length(obj@name) != 1)
     "Gene name should be a single string"
  else if (!nzchar(obj@name))
     "name is empty"
  else
     TRUE
})

## NOTE: arguments in definition for validity method for class 'Gene'
changed from (obj) to (object)

## Class "Gene" [in ".GlobalEnv"]
##
## Slots:
##
## Name:        name
## Class: character
```

Invalid object instantiation

```
new("Gene", name = "")

## Error: invalid class "Gene" object: name is empty

new("Gene", name = c("TP53", "EGFR"))

## Error: invalid class "Gene" object: Gene name
should be a single string
```

Creating New Object from Old

initialize function arguments

Object an old S4 object.

... typically, new slot values.

Creation via initialize

```
gene1 = new("Gene", name = "ZNF750")
gene2 = initialize(gene1, name = "FAT1")
gene2

## An object of class "Gene"
## Slot "name":
## [1] "FAT1"
```

Method Creation

setMethod function arguments

f	the generic function name.
signature	the mapping of arguments in dispatch signature to classes.
definition	the method definition.

Method Creation: Example 1

```
 setMethod("print", c("x"="Gene"),
 function(x){
   print(paste("The gene name is", x@name, sep=" "))
})

## Creating a generic function for 'print' from package 'base' in the
global environment

## [1] "print"

 print(gene1)

## [1] "The gene name is ZNF750"

 print(gene2)

## [1] "The gene name is FAT1"
```

Method Creation: Example 2

```
 setGeneric("showName", signature = "x",
    function(x) standardGeneric("showName"))
```

```
## [1] "showName"
```

```
 setMethod("showName", c("x"="Gene"),
 function(x){
   print(paste("The gene name is", x@name, sep=" "))
})
```

```
## [1] "showName"
```

```
 showName(gene1)
```

```
## [1] "The gene name is ZNF750"
```

Coerce

setAs function arguments; produces coerce methods

- from the old object's class.
- to the new object's class.
- def the method definition.

Coerce: Example

```
 setAs("Gene","character",
 function(from) from@name)
 as(gene2, "character")

## [1] "FAT1"
```

ExpressionSet class

ExpressionSet

The ExpressionSet class is designed to combine several different sources of information into a single convenient structure. An ExpressionSet can be manipulated (e.g., subsetted, copied) conveniently, and is the input or output from many Bioconductor functions.

ExpressionSet: Slots

```
library(Biobase)
isVirtualClass("ExpressionSet")

## [1] FALSE

getSlots("ExpressionSet")

##       experimentData            assayData            phenoData
##              "MIAME"          "AssayData" "AnnotatedDataFrame"
##          featureData           annotation         protocolData
## "AnnotatedDataFrame"          "character" "AnnotatedDataFrame"
##    .__classVersion__
##           "Versions"
```

ExpressionSet: Methods

```
getMethods("ExpressionSet")
```

```
## An object of class "MethodsList"
## Slot "methods":
## $environment
## function (assayData, phenoData = annotatedDataFrameFrom(assayData,
##     byrow = FALSE), featureData = annotatedDataFrameFrom(assayData,
##     byrow = TRUE), experimentData = MIAME(), annotation = character(),
##     protocolData = annotatedDataFrameFrom(assayData, byrow = FALSE),
##     ...)
## {
##     .ExpressionSet(assayData = assayData, phenoData = phenoData,
##         featureData = featureData, experimentData = experimentData,
##         annotation = annotation, protocolData = protocolData,
##         ...)
## }
## <environment: namespace:Biobase>
## attr(,"target")
##     assayData
## "environment"
## attr(,"defined")
##     assayData
## "environment"
## attr(,"generic")
## [1] "ExpressionSet"
## attr(,"generic")attr(,"package")
## [1] "Biobase"
## attr(,"class")
## [1] "MethodDefinition"
```

References

- http://www.bioconductor.org/help/course-materials/2010/AdvancedR/S4InBioconductor.pdf
- http://adv-r.had.co.nz/OO-essentials.html
- John M. Chambers. Software for Data Analysis: Programming with R. Springer, New York, 2008. ISBN-13 978-0387759357.
- http://developer.r-project.org/howMethodsWork.pdf
- http://www.ci.tuwien.ac.at/Conferences/useR-2004/Keynotes/Leisch.pdf
- https://www.stat.auckland.ac.nz/S-Workshop/Gentleman/S4Objects.pdf

Next

1 Bioconductor

2 S4 in Bioconductor

3 Genetic Variants Annotation in Bioconductor
- Data and Packages
- Analysis
- Reference

4 Biological Data Visualization: ggbio

Data and Packages

- VariantAnnotation
- cgdv17
- org.Hs.eg.db
- TxDb.Hsapiens.UCSC.hg19.knownGene
- BSgenome.Hsapiens.UCSC.hg19
- PolyPhen.Hsapiens.dbSNP131

Installtion

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("VariantAnnotation",
"cgdv17",
"org.Hs.eg.db",
"TxDb.Hsapiens.UCSC.hg19.knownGene",
"BSgenome.Hsapiens.UCSC.hg19",
"PolyPhen.Hsapiens.dbSNP131"))
```

Load VCF into R

```
library(VariantAnnotation)
library(cgdv17)
file <- system.file("vcf", "NA06985_17.vcf.gz", package = "cgdv17")
hdr <- scanVcfHeader(file)
info(hdr)
```

```
## DataFrame with 3 rows and 3 columns
##         Number        Type                 Description
##    <character> <character>                 <character>
## NS           1     Integer Number of Samples With Data
## DP           1     Integer                 Total Depth
## DB           0        Flag dbSNP membership, build 131
```

Load VCF into R

```
geno(hdr)
```

```
## DataFrame with 12 rows and 3 columns
##              Number        Type                                Description
##         <character> <character>                                <character>
## GT                1      String                                   Genotype
## GQ                1     Integer                          Genotype Quality
## DP                1     Integer                                Read Depth
## HDP               2     Integer                       Haplotype Read Depth
## HQ                2     Integer                          Haplotype Quality
## ...             ...         ...                                        ...
## mRNA              .      String                           Overlaping mRNA
## rmsk              .      String                         Overlaping Repeats
## segDup            .      String Overlaping segmentation duplication
## rCov              1       Float                          relative Coverage
## cPd               1      String                       called Ploidy(level)
```

Get Gene ID

```
library(org.Hs.eg.db)
genesym <- c("TRPV1", "TRPV2", "TRPV3")
geneid <- select(org.Hs.eg.db, keys = genesym,
keytype = "SYMBOL", columns = "ENTREZID")
geneid
```

```
##   SYMBOL ENTREZID
## 1  TRPV1     7442
## 2  TRPV2    51393
## 3  TRPV3   162514
```

Create a transcripts list by gene

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
txbygene = transcriptsBy(txdb, "gene")
tx_chr17 <- keepSeqlevels(txbygene, "chr17")
tx_17 <- renameSeqlevels(tx_chr17, c(chr17 = "17"))
# Create the gene ranges for the TRPV genes
rngs <- lapply(geneid$ENTREZID, function(id) range(tx_17[names(tx_17) %in
% id]))
gnrng <- unlist(do.call(c, rngs), use.names = FALSE)
names(gnrng) <- geneid$SYMBOL
```

Subset VCF file

```
param <- ScanVcfParam(which = gnrng, info = "DP", geno = c("GT", "cPd"))
vcf <- readVcf(file, "hg19", param)
vcf
```

```
## class: CollapsedVCF
## dim: 405 1
## rowData(vcf):
##   GRanges with 5 metadata columns: paramRangeID, REF, ALT, QUAL, FILTER
## info(vcf):
##   DataFrame with 1 column: DP
## info(header(vcf)):
##       Number Type    Description
##    DP 1      Integer Total Depth
## geno(vcf):
##   SimpleList of length 2: cPd, GT
## geno(header(vcf)):
##        Number Type   Description
##    cPd 1      String called Ploidy(level)
##    GT  1      String Genotype
```

Subset VCF

```
head(fixed(vcf))
```

```
## DataFrame with 6 rows and 4 columns
##               REF                     ALT      QUAL      FILTER
##   <DNAStringSet> <DNAStringSetList> <numeric> <character>
## 1              A                       G       120        PASS
## 2              A                               0        PASS
## 3          AAAAA                               0        PASS
## 4             AA                               0        PASS
## 5              C                       T        59        PASS
## 6              T                       C       157        PASS
```

Subset VCF

```
geno(vcf)

## List of length 2
## names(2): cPd GT
```

```
seqlevels(vcf)

## [1] "17"

head(seqlevels(txdb))

## [1] "chr1" "chr2" "chr3" "chr4" "chr5" "chr6"

intersect(seqlevels(vcf), seqlevels(txdb))

## character(0)

vcf_mod <- renameSeqlevels(vcf, c(`17` = "chr17"))
intersect(seqlevels(vcf_mod), seqlevels(txdb))

## [1] "chr17"
```

Get Variants Info

```
cds <- locateVariants(vcf_mod, txdb, CodingVariants())
five <- locateVariants(vcf_mod, txdb, FiveUTRVariants())
splice <- locateVariants(vcf_mod, txdb, SpliceSiteVariants()
intron <- locateVariants(vcf_mod, txdb, IntronVariants())
all <- locateVariants(vcf_mod, txdb, AllVariants())

## Warning: trimmed start values to be positive
## Warning: trimmed end values to be <= seqlengths
```

variants and gene

```
## Did any variants match more than one gene
table(sapply(split(values(all)[["GENEID"]], values(all)
[["QUERYID"]]), function(x) length(unique(x)) >
    1))

##
## FALSE  TRUE
##   367    38
```

Summarize variant number by gene

```
idx <- sapply(split(values(all)[["QUERYID"]], values(all)[["GENEID"]]), unique)
sapply(idx, length)

## 125144 162514  23729  51393   7442  84690
##      1    196      2     63    146     35
```

Summarize variant location by gene

```
sapply(names(idx), function(nm) {
    d <- all[values(all)[["GENEID"]] %in% nm, c("QUERYID", "LOCATION")]
    table(values(d)[["LOCATION"]][duplicated(d) == FALSE])
})
```

```
##             125144 162514 23729 51393 7442 84690
## spliceSite       0      2     0     0    1     0
## intron           0    153     0    58  117    19
## fiveUTR          0      2     0     1    3     5
## threeUTR         0     24     2     1    2     0
## coding           0      5     0     3    8     0
## intergenic       0      0     0     0    0     0
## promoter         1     10     0     0   15    11
```

Reference

- Using Bioconductor to Annotate Genetic Variants: http://www.bioconductor.org/help/workflows/variants/

Next

1. Bioconductor
2. S4 in Bioconductor
3. Genetic Variants Annotation in Bioconductor
4. Biological Data Visualization: ggbio
   - Overview and Installation
   - Study Cases