Laboratorio 2 Bioinformática - Bioconductor

Manuel Villalobos Cid

26 de septiembre de 2016

1. Introducción

Bioconductor es un proyecto de código abierto para análisis de datos de genómica (secuencias, microarreglos, anotaciones y otros tipos de datos). Su funcionamiento está orientado principalmente para R, sin embargo, es posible su utilización en otros lenguajes de programación como Python y herramientas como MATLAB.

El proyecto se originó el año 2001, supervisado por un equipo del Centro de Investigación del Cancer Fred Hutchinson. Hasta el año 2016, se dispone de **1024** paquetes específicos de aplicación. Éstos son actualizados semestralmente.

El objetivo de este laboratorio consiste introducir al alumno a la utilización de Bioconductor en forma general, utilizando ejemplos específicos de aplicación. Debido a que Bioconductor dispone de un elevado número de librerías y su respectiva documentación, es necesario que el alumno estudie de manera particular los requerimientos a utilizar en trabajos futuros.

2. Instalación de Bioconductor

La instalación de Bioconductor requiere utilizar las siguientes sentencias. El proceso requiere de aproximadamente 10 minutos de descarga.

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

Para instalar un paquete específico, como por ejemplo el paquete ALL ("Data of T- and B-cell Acute Lymphocytic Leukemia from the Ritz Laboratory at the DFCI"), es necesario usar la siguiente sentencia:

```
source("http://www.bioconductor.org/biocLite.R")
biocLite("ALL")
```

3 Manipulación de secuencias

Las base de datos de secuencias almacenan datos de ADN y productos génicos. Ejemplos de ellas son: EMBL, Genebank y DDBJ. La mayoría dispone de secuencias en formato de texto plano, siendo los formatos GeneBank, EMBL y FASTA los más utilizados.

Para manejo eficiente de cadenas de texto, efectuar comparación de secuencias y manipular secuencias de gran largo, **Bioconductor** presenta una libería denominada **Biostring**.

3.1 Instalación de Biostring

- Para instalar el paquete que incluye la librería **Biostring** se debe utilizar la siguiente sentencia:

```
#biocLite(Biostrings)
library(Biostrings)
```

```
## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, as.vector, cbind,
##     colnames, do.call, duplicated, eval, evalq, Filter, Find, get,
##     grep, grepl, intersect, is.unsorted, lapply, lengths, Map,
##     mapply, match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, Position, rank, rbind, Reduce, rownames, sapply,
##     setdiff, sort, table, tapply, union, unique, unlist, unsplit

## Loading required package: S4Vectors

## Loading required package: stats4

## Loading required package: IRanges

## Loading required package: XVector
```

3.2 Generación y manipulación de secuencias

- Biostring disponde de tres clases denominadas **DNAString**, **RNAString** y **AAString**. Éstas permiten almacenar una secuencia de ADN, ARN y aminoácidos como una cadena de texto. Cree una secuencia de ADN siguiendo el ejemplo propuesto a continuación:

```
dnaSeq = DNAString("TTCAGATCTAGTTCGTGTGTGACTGATGATCTGTCACACGTTTTTCTGATCTTCTGACTAGTCGAT")
```

- Puede acceder a los elementos de una secuencia como si se tratase de un vector. Sin embargo, debido al tamaño de las secuencias reales, se recomienda utilizar la función **substr**. Acceda a las primeros 10 caracteres de su secuencia utilizando ambos métodos:

```
dnaSeq[1:10]
```

```
##   10-letter "DNAString" instance
## seq: TTCAGATCTA
```

```
substr(dnaSeq,1,10)
```

```
##   10-letter "DNAString" instance
## seq: TTCAGATCTA
```

- Para conocer características como el largo de la secuencia, obtener el la cadena complementaria y la frecuencia de las bases, es posible utilizar las siguientes sentencias:

```
length(dnaSeq)                #Establece el largo de la secuencia
```

```
## [1] 66
```

```
reverseComplement(dnaSeq)     #Genera una cadena complementaria reversa
```

```
##   66-letter "DNAString" instance
## seq: ATCGACTAGTCAGAAGATCAGAAAAACGTGTGACAGATCATCAGTCACACACGAACTAGATCTGAA
```

```
alphabetFrequency(dnaSeq)     #Indica la cardinalidad de cada caracter
```

```
##  A  C  G  T  M  R  W  S  Y  K  V  H  D  B  N  -  +  .
## 12 13 14 27  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

- Para comparar secuencias se puede utilizar la funciÃ³n **compareString**, utilice esta función basándose en el siguiente ejemplo:

```
compareStrings(dnaSeq,dnaSeq)
```

```
## [1] "TTCAGATCTAGTTCGTGTGTGACTGATGATCTGTCACACGTTTTTCTGATCTTCTGACTAGTCGAT"
```

```
dnaSeq2 = DNAString("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA")
compareStrings(dnaSeq,dnaSeq2)
```

```
## [1] "???A?A???A???????????A???A??A??????A?A???????????A???????A??A????A?"
```

3.3 Archivos FASTA

El formato FASTA se utiliza para representar secuencias de Ácidos nucléicos y péptidos en forma de cadenas de textos empleando un caracter por cada monómero.

Una secuencia en formato FASTA comienza con una descripción de la secuencias (>), seguido por los datos de la misma. El tipo de encabezado depende de cada estÃ¡ndar.

> gi|129295|sp|P01013|OVAX_CHICK GENE X PROTEIN(OVALBUMIN-RELATED)

QIKDLLVSSSTDLDTTLVLVNAIYFKGMWKTAF NAEDTREMPFHVTKQESKPVQMMCMNNSFNVAT

LPAEKMKILELPFASGDLSMLVLLPDEVSDLER IEKTINFEKLTEWTNPNTMEKRRVKVYLPQMK

EEKYNLTSVLMALGMTDLFIPSANLTGISSAES LKISQAVHGAFMELSEDGIEMAGSTGVIEDIKH

SPESEQFRADHPFLFLIKHNPTNTIVYFGRYWS

- Para leer un archivo de este tipo de formato se puede utilizar la siguiente sentencia. El archivo debe estar ubicado en la carpeta de trabajo.

```
dnaSeq = readDNAStringSet("primates_14.fasta")
```

- Aplique las funciones aplicadas en la sección anterior a esta nueva secuencia.

```
names(dnaSeq) #Conocer el nombre de las especies
```

```
##  [1] "Mouse"     "Bovine"    "Lemur"     "Tarsier"   "Squir_Mon"
##  [6] "Jpn_Macaq" "Rhesus_Ma" "Crab-E.Ma" "BarbMacaq" "Gibbon"
## [11] "Orang"     "Gorilla"   "Chimp"     "Human"
```

```
names(dnaSeq[1]) #Reconocer una especie
```

```
## [1] "Mouse"
```

```
length(dnaSeq) #Saber cuantas especies son
```

```
## [1] 14
```

```
reverseComplement(dnaSeq[1]) #Complemento
```

```
##   A DNAStringSet instance of length 1
##     width seq                                               names
## [1]   232 GTTATTTATGGTGCGTATATT...TGTTTGGATGTTTTTTTGGT Mouse
```

```
alphabetFrequency(dnaSeq) #Contador de caracteres
```

```
##         A   C  G  T M R W S Y K V H D B N - + .
##  [1,] 102  73  7 50 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [2,] 103  79  6 44 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [3,]  93  83  5 51 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [4,]  97  73  8 54 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [5,]  95  75 13 48 0 0 0 0 0 0 0 0 0 0 0 1 0 0
##  [6,]  82  96 11 43 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [7,]  85  97  9 41 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [8,]  79  95 13 45 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [9,]  90  92  6 44 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [10,]  77 104 15 36 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [11,]  79 117  7 29 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [12,]  84 107  6 35 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [13,]  76 105 10 41 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [14,]  75 108 11 38 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
dinucleotideFrequency(dnaSeq) #Contador de combinaciones
```

```
##       AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
##  [1,] 43 33  4 22 31 24  3 14  2  5  0  0 25 11  0 14
##  [2,] 40 36  3 24 36 30  2 10  3  1  0  2 23 12  1  8
##  [3,] 39 34  2 18 32 24  1 25  0  4  0  1 21 21  2  7
##  [4,] 39 26  4 28 30 23  3 16  2  5  0  1 25 19  1  9
##  [5,] 41 27  4 22 27 28  5 14  5  4  0  4 20 16  4  8
##  [6,] 32 31  4 15 32 37  2 24  2  7  1  1 15 21  4  3
##  [7,] 37 31  1 16 34 39  6 17  1  5  1  2 12 22  1  6
##  [8,] 32 34  2 11 26 39  7 22  2  7  1  3 18 15  3  9
##  [9,] 37 33  3 17 38 37  3 13  0  5  0  1 14 17  0 13
## [10,] 20 39  8 10 37 41  6 19  5 10  0  0 14 14  1  7
## [11,] 30 38  2  9 37 59  4 16  1  4  0  2 10 16  1  2
## [12,] 31 35  1 17 40 51  3 12  1  5  0  0 11 16  2  6
## [13,] 24 37  2 13 40 40  7 17  3  5  0  2  8 23  1  9
## [14,] 24 39  2 10 38 41  8 20  2  7  0  2 10 21  1  6
```

3.4 Alineamiento de secuencias

Esta sección valida los alineamientos planteados durante la clase.

3.4.1 Secuencia común mas corta

```
s1 =  DNAString("ATCTGAT")
s2 = DNAString("TGCATA")
mat = nucleotideSubstitutionMatrix(match = 1, mismatch = -1, baseOnly = TRUE)
globalAlign =   pairwiseAlignment(s1, s2, type="global",
substitutionMatrix = mat,gapOpening=0, gapExtension = 0)
print(globalAlign)
```

```
## Global PairwiseAlignmentsSingleSubject (1 of 1)
## pattern: [2] T-C-TGA
## subject: [1] TGCAT-A
## score: 4
```

3.4.2 Alineamiento global

```
s1 =  DNAString("ATCTGAT")
s2 = DNAString("TGCATA")
mat = nucleotideSubstitutionMatrix(match = 1, mismatch = -1, baseOnly = TRUE)
globalAlign =   pairwiseAlignment(s1, s2, type="global",
substitutionMatrix = mat,gapOpening=0, gapExtension = -1)
print(globalAlign)
```

```
## Global PairwiseAlignmentsSingleSubject (1 of 1)
## pattern: [2] T-C-TGA
## subject: [1] TGCAT-A
## score: -1
```

3.4.3 Alineamiento local

```
s1 =  DNAString("ATCTGAT")
s2 = DNAString("TGCATA")
mat = nucleotideSubstitutionMatrix(match = 1, mismatch = -1, baseOnly = TRUE)
globalAlign =   pairwiseAlignment(s1, s2, type="local",
                                  substitutionMatrix = mat,gapOpening=0, gapExtension = -1)
print(globalAlign)
```

```
## Local PairwiseAlignmentsSingleSubject (1 of 1)
## pattern: [4] TG-AT
## subject: [1] TGCAT
## score: 3
```

3.5 Ejemplo de aplicacion: árbol filogenético

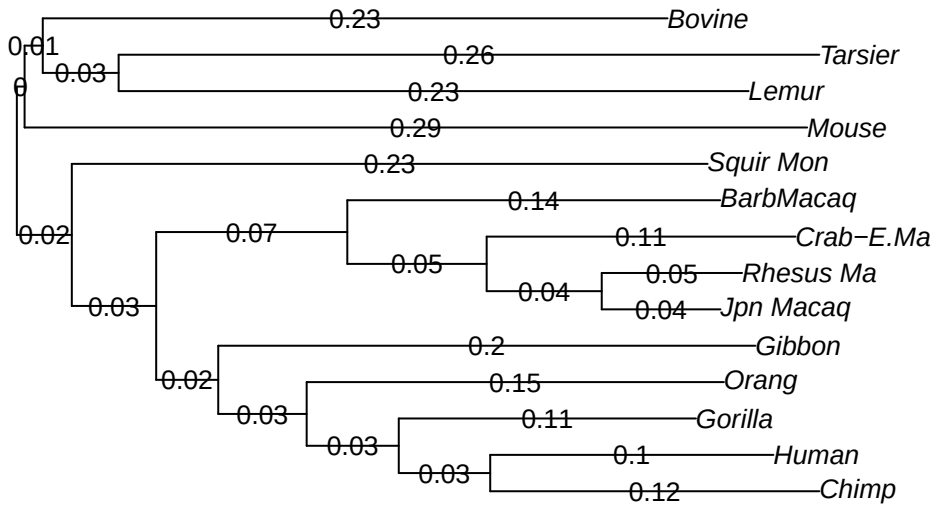
El siguiente ejemplo muestra como representar las relaciones evolutivas entre un conjunto de especies, considerando el alineamiento múltiple un determinado marcador génico. Para ello, se utilizará como ejemplo el conjunto de 14 primates y como método de reconstrucción ***Neighbour Joining*** .

```
#install.packages("phangorn")
library("phangorn")
```

```
## Warning: package 'phangorn' was built under R version 3.2.5
```

```
## Loading required package: ape
```

```
dnaSeq = read.phyDat("primates_14.fasta", format="fasta", type="DNA")
mat_dist = dist.hamming(dnaSeq)
tree = midpoint(NJ(mat_dist))
plot(tree,"phylogram",cex=0.8,label.offset = 0)
edgelabels(round(tree$edge.length,2), adj = c(0.5, 0.5), frame = "none",
           pch = NULL, thermo = NULL, pie = NULL, piecol = NULL,
           col = "black", bg = "white", cex = 0.8)
```



3.6 Actividad propuesta

Como primer acercamiento a la comparación de secuencias, averigue en qué consiste la función **CompareString** y el significado de los símbolos presentado en sus resultados: ?, +, -.