R3CPET user manual

Mohamed Nadhir Djekidel, Yang Chen et al

nde12@mails.tsinghua.edu.cn

April 7, 2020

Contents

1 Introduction

The breakthrough in chromatin conformation study done in the last decade has shed some light on many aspects of gene transcription mechanisms and revealed that chromatin crosstalk plays an important role in connecting regulatory sequences to their target promoters through loop formation

However, a key limitation of the existing chromatin conformation assays is that they can just give us the genomic coordinates of the interacting DNA fragments but don't tell us on the proteins involved.One used solution is to use *Chip-Seq* to get the list of the proteins involved at these borders but one limitation is that not all the proteins can be captured and some don't have a specific anti-body.

Thus, computational methods are still useful to give some insight on the protein candidates that can play a role in maintaining these interactions. *R3CPET* comes as a tool to fill this gap and try to infer the loop-maintaining network(s) in a more concise manner.

3CPET is based on the following idea : if we can have the list of the protein networks that maintain all of the DNA-interactions, then we can infer the set the most enriched networks. One of the widely used statistical model in this kind of problems is the HLDA model which is a non-parametric Bayesian that enables us to infer the number of different groups from the data.

In order to apply HLDA, we need to have a corpus of documents and we assign each word to a topic. In our case each document is a network and each word is an edge in that network. To create a network for each interaction we use *(i)* find the set of the proteins involved a the boundaries of a loop and this by using *Chip-Seq* peaks or *motifs* data.

In the next step, we use the information in a background PPI to construct the network connecting the two interacting DNA fragments.

The networks are then converted into a *bag* of edges (*i.e* allow repetition) and fed to the HLDA algorithm.

In this model, we suppose that each network $(j_n)_{n=1}^{\infty}$ is made-up of a mixture of protein complexes each with different a proportion $\theta_n \sim DP(\alpha, \pi)$. To infer the number of clusters, the model suppose that we have an infinite number of chromatin maintainer networks distributed according to a certain distribution H. To enable the sharing the complexes (β_k) across all the networks, an intermediate discrete distribution is introduced $(\beta_k)_{k=1}^{N}$ sampled from the base distribution H using a stick breaking construction $\pi|\gamma \sim GEM(\gamma)$.

Figure 1: Illustration of the network construction procedure

Figure 2: HLDA model

2 Using *R3CPET*

R3CPET package were built to enable the user to upload the data step-by-step and thus giving him a more granular control of the data that he uses. Maybe it sounds like the use needs more labor, but the user can create an `S3` wrapper function that encapsulate the workflow functions into one method.

Four main classes are provided by *R3CPET*:

1. `ChiapetExperimentData` - a container for the raw data to use. 3 types of data can be loaded into this class respectively :*ChIA-PET* interactions, *ChIP-Seq* peaks and the background *PPI*.
2. `NetworkCollection` - Holds the list of the build networks for each DNA-interaction and their information.
3. `HLDAResult` - contains the results of the HLDA algorithm.
4. `ChromMaintainers` - contains the final results after processing the HLDA results (*i.e*: list of networks and their nodes).

In addition to these classes additional helper methods for GO enrichment and and gene conversion.

2.1 Loading data

Before starting the analysis different kinds of dataset need be loaded. This can be done using a `ChiapetExperimentData` object.

The user can load the data using the `ChiapetExperimentData` constructor. By passing: ChIA-PET interactions data, transcription factors binding site (TFBS) and a protein-protein interaction network. The ChIA-PET interactions can be passed as path to a file or a `GRanges` object. The same is for the TFBS.

```
> library(R3CPET)
> petFile <- file.path(system.file("example", package = "R3CPET"),
+     "HepG2_interactions.txt")
> tfbsFile <- file.path(system.file("example", package = "R3CPET"),
+     "HepG2_TF.txt.gz")
> x <- ChiapetExperimentData(pet = petFile, tfbs = tfbsFile, IsBed = FALSE,
+     ppiType = "HPRD", filter = TRUE)
```

Three types of data can be loaded :

- *ChIA-PET interactions* - which can have two formats: (i) the first type is a file in which the first six columns indicate the left and right interacting parts (generally parsed from the ChIA-PET tool). For example

```
> petPath <- system.file("example", "HepG2_interactions.txt", package = "R3CPET")
> petFile <- read.table(petPath, sep = "\t", header = TRUE)
> head(petFile)

##   chromleft startleft endleft chromright startright
## 1      chr1   1282738 1283678       chr1    1283743
## 2      chr1   1370817 1371926       chr1    1372332
## 3      chr1   2159841 2161268       chr1    2161279
```

```
## 4      chr1   2159841 2161268       chr1    2162130
## 5      chr1   2161279 2162108       chr1    2162130
## 6      chr1   9187131 9188440       chr1    9188469
##   endright counts       pvalue  qvalue
## 1  1284926      6 1.075586e-21 8.8e-05
## 2  1373312      8 3.536936e-26 8.8e-05
## 3  2162108     18 5.487730e-57 8.8e-05
## 4  2163908      6 8.048577e-17 8.8e-05
## 5  2163908     17 2.792212e-55 8.8e-05
## 6  9189622      7 2.576820e-28 8.8e-05
```

Here only the 6 first columns are considered. It is up to the user to filter the significant interactions for him.

The second type of files that can be loaded is a four columns file in which the first three columns indicate the genomic location of a DNA region and the forth column indicate if the region is located at the right or left side. The IDs in the fourth column should have the pattern `PET#\\d+\\.1` for the left side and `PET#\\d+\\.2` for the right side. if the number of the left side interactions is different from the right side an error will be raised.

```
> petPath <- system.file("example", "HepG2_centered.bed", package = "R3CPET")
> petFile <- read.table(petPath, sep = "\t", header = FALSE, comment.char = "+")
>
> head(petFile)

##     V1      V2      V3      V4
## 1 chr1 1241234 1242234 PET#1.1
## 2 chr1 1242724 1243724 PET#1.2
## 3 chr1 1282708 1283708 PET#2.1
## 4 chr1 1283834 1284834 PET#2.2
## 5 chr1 1370871 1371871 PET#3.1
## 6 chr1 1372322 1373322 PET#3.2
```

The method `loadPETs` can be used to load the data

```
> ## if it has 6 columns format IsBed = FALSE
> petPath <- system.file("example", "HepG2_interactions.txt", package = "R3CPET")
> x <- loadPETs(x, petFile = petPath, IsBed = FALSE)

## 304 interacting DNA regions loaded
```

if the file is 4 columns BED file you can set `IsBed = TRUE`

```
> ## loading a 4 columns BED file
> petPath <- system.file("example", "HepG2_centered.bed", package = "R3CPET")
> x <- loadPETs(x, petFile = petPath, IsBed = TRUE, header = FALSE)
```

The `pet(x)` accessor method can be used to read the loaded interactions as *GRanges* object.

- *ChIP-Seq peaks* - All the *ChIP-Seq* peaks of the different TF should be merged into a 4 columns in which the first 3 columns indicate the position of the peak and the last column indicate the associated TF. The `loadTFBS` method can be used to do so.

```
> ## loading a 4 columns BED file
> TFPath <- system.file("example", "HepG2_TF.txt.gz", package = "R3CPET")
```

```
> TF <- read.table(TFPath, sep = "\t", header = FALSE)
> head(TF)

##      V1       V2       V3    V4
## 1  chr8 67804615 67804862  HEY1
## 2 chr15 52200521 52200848 FOSL2
## 3 chr15 70820640 70820932 EP300
## 4 chr13 48775200 48775538 EP300
## 5  chr3 23987219 23988231 FOSL2
## 6  chr6 56263941 56265486 FOSL2

> x <- loadTFBS(x, tfbsFile = TFPath)

## a total of  81414 binding sites for 6 TF  were loaded
```

- *Protein interaction* - a background `PPI` is needed to do the networks construction. The package comes with two build in `PPI` : the `HPRD` and the `Biogrid PPI`.

```
> data(HPRD)
> data(Biogrid)
>
> PPI.HPRD

## IGRAPH 84de311 UN-- 9616 39042 --
## + attr: name (v/c)
## + edges from 84de311 (vertex names):
##  [1] ALDH1A1--ALDH1A1 ITGA7  --CHRNA1  PPP1R9A--ACTG1
##  [4] SRGN   --CD44    GRB7   --ERBB2   ERBB2  --PAK1
##  [7] ERBB2  --DLG4    ERBB2  --PIK3R2  ERBB2  --PTPN18
## [10] ERBB2  --ERBB2IP SMURF2 --ARHGAP5 ERBB2  --NF2
## [13] ERBB2  --CD82    ERBB2  --ERRFI1  CD44   --MMP7
## [16] ERBB2  --TOB1    ERBB2  --MUC4    ERBB2  --PICK1
## [19] SMURF2 --TXNIP   DDX20  --ETV3    TLE1   --FOXG1
## [22] FOXG1  --TLE3    FOXG1  --HDAC1   FOXG1  --SMAD1
## + ... omitted several edges

> PPI.Biogrid

## IGRAPH 2eeef88 UN-- 16227 169166 --
## + attr: name (v/c)
## + edges from 2eeef88 (vertex names):
##  [1] MAP2K4--FLNC     MYPN  --ACTN2    ACVR1 --FNTA
##  [4] GATA2 --PML      RPA2  --STAT3    ARF1  --GGA3
##  [7] ARF3  --ARFIP2   ARF3  --ARFIP1   AKR1A1--EXOSC4
## [10] XRN1  --ALDOA    APP   --APPBP2   APLP1 --DAB1
## [13] CITED2--TFAP2A   TFAP2A--EP300    APOB  --MTTP
## [16] ARRB2 --RALGDS   CSF1R --GRB2     GRB2  --PRRC2A
## [19] LSM1  --NARS     SLC4A1--SLC4A1AP BCL3  --BARD1
## [22] ADRB1 --GIPC1    BRCA1 --ATF1     BRCA1 --MSH2
## + ... omitted several edges
```

The `loadPPI` method can be used to load and filter the `PPI` according to different criteria.

```
> loadPPI(object, type = c("HPRD", "Biogid"), customPPI = NULL,
+     filter = FALSE, term = "GO:0005634", annot = NULL, RPKM = NULL,
+     threshold = 1)
```

if `customPPI` one of the built-in `PPI` will be used otherwise the user can provide a path to an `ncol` formatted graph (two column to indicate interacting nodes) or directly provide an *igraph* object.

The user can also do some filtering to remove proteins that he thinks are not significant. by default the package keeps only the proteins that are located at nucleus (`term = "GO:0005634"`). The user can provide his own annotation to the `annot` parameter. In some cases the user want to keep only the cell specific genes, thus he can filter genes according to their gene expression by providing a two columns gene expression table to the `RPKM` parameter and set the `threshold`.

```
> ## loading the PPI with GO filtering
> x <- loadPPI(x, type = "HPRD", filter = TRUE)

## loading HPRD network, with 9616 nodes
```

2.2 Creating indexes

When the user thinks that the data that he loaded is ok, he need to create indexes for processing in the further steps. Indexes can be created by calling the method `createIndexes`.

```
> x <- createIndexes(x)

## [1] "Creating PET table"
## [1] "Sorting Table"
## [1] "Get TF associated with each PETs"
## [1] "Creating Motifs table"
## [1] "Sorting Table"
## [1] "Creating hasMotif table"
## [1] "Sorting Table"

> x

## class: ChiapetExperimentData
## 304  interacting regions
## 6 TF used
## Background PPI:
##  nodes: 3447   edges: 15829
## indexes tables have been created
```

2.3 building networks for DNA regions

One all the data are loaded we can go to the next step and build the protein interaction networks for each chromatin loop. The `buildNetworks` is used and a `NetworkCollection` is returned. When the networks are built, the rare and the more frequent edges are removed as they are considered to be not specific. By default, edges that appear in less than 25% or more than 75% of the total networks are removed.

the user can set these values by modifying the `minFreq` and `maxFreq` parameters of the `buildNetworks` method.

```
> nets <- buildNetworks(x, minFreq = 0.1, maxFreq = 0.9)
> nets
```

```
## class NetworkCollection
## 145 network loaded
## 128 diffrent edges has been used
```

The `buildNetworks` uses the *parallel* package to parallelize the processing.If the user has 4 cores, then 4 *R*instances will be lunched at the background each running on a core and handling part of the data.

2.4 Inferring chromatin maintainer networks

At this step the `InferNetworks` can be used to run the HLDA algorithm and infer the set of the most enriched chromatin maintainer networks. by default, the algorithm do a maximum of 500 iteration or stops after 1 hour.

```
InferNetworks(object, thr = 0.5, max_iter = 500L, max_time = 3600L,
    eta = 0.01, gamma = 1, alpha = 1)
```

To control the behaviour of the algorithm, users can modify the values of the HDP algorithm paramters `eta`, `gamma` and `alpha`. Briefly, `eta` and `alpha` control the sparcity of the edge-to-CMN matrix. In other words, smaller values allow force an edge to belong to a small number of CMNs (ideally one) and larger values (>1) alow them to be uniformly assigned. `gamma` controls the number of CMNs, smaller values leads to less prediced CMNs while larger values leads to more CMNs. Users can check figures S13-S16 of the mauscript.

When the algorithm finishes we get a matrix that indicates the degree of partnership of each edge to the inferred networks (each network is distribution over edges). Thus, to get the top elements in each network, we use the parameter `thr` to select the edges that capture `thr`% of the network. Thus, more general networks tend to have bigger number elements and more specific networks have less elements. At the end a `ChromMaintainers` object is returned.

```
> hlda <- InferNetworks(nets)
> hlda
```

```
## class: ChromMaintainers
## HLDA Results:
## ------------
## 145 element have been classified into 10 topics
## Number of different words 128
```

The different slots of the `hlda` object can be accessed using the following accessor methods:

- `topEdges` : to get the list of the top edges in each network.
- `topNodes` : to get the list of the top nodes in each network.
- `networks` : to get a list `igraph` objects.

```
> head(topEdges(hlda))
```

```
##      Topic_1         Topic_2          Topic_3
## [1,] "EP300_MYOD1"   "ESR1_ESRRA"     "FOSL2_JUND"
## [2,] "ARNT_EP300"    "ESR1_NCOA1"     "FOSL2_JUN"
```

```
## [3,] "GABPA_SP1"    "ESRRA_NCOA1"    "JUN_MYOD1"
## [4,] "CREBBP_SMAD3" "NCOA1_PPARGC1A" "JUN_SMAD3"
## [5,] "MYOD1_SP1"    "ESR1_PPARGC1A"  "FOSL2_JUNB"
## [6,] "CREBBP_MYOD1" "ESR1_SMAD3"     ""
##      Topic_4       Topic_5       Topic_6       Topic_7
## [1,] "GABPA_SP1"   "ESR1_JUND"   "JUND_SMAD4"  "FOSL2_JUND"
## [2,] "MAPK1_SP1"   "EP300_ESR1"  "BRCA1_SMAD4" "EP300_JUN"
## [3,] "MAPK1_SMAD4" "EP300_SMAD4" "BCL6_EP300"  "FOSL2_JUN"
## [4,] "MAPK1_SMAD3" "ESR1_SMAD4"  "ESR1_SMAD3"  "JUN_SMAD3"
## [5,] "JUND_MAPK1"  "EP300_JDP2"  "BRCA1_SMAD3" ""
## [6,] "BCL6_MAPK1"  ""            ""            ""
##      Topic_8        Topic_9       Topic_10
## [1,] "EP300_GABPA"  "EP300_NCOA1" "CREBBP_PROX1"
## [2,] "EP300_SP1"    "GTF2B_NCOA1" "MAPK1_PRKCD"
## [3,] "CREBBP_EP300" "ESRRA_GTF2B" "MAPK1_NCOA3"
## [4,] "ATF1_CREBBP"  "ESR1_NCOA3"  "MAPK1_NCOA1"
## [5,] "ATF1_GABPA"   "ESRRA_NCOA3" ""
## [6,] "CREBBP_GABPA" "EP300_ESR1"  ""
```

```
> head(topNodes(hlda))
```

```
##      Topic_1  Topic_2    Topic_3 Topic_4 Topic_5 Topic_6
## [1,] "EP300"  "ESR1"     "FOSL2" "GABPA" "ESR1"  "JUND"
## [2,] "MYOD1"  "ESRRA"    "JUND"  "SP1"   "JUND"  "SMAD4"
## [3,] "ARNT"   "NCOA1"    "JUN"   "MAPK1" "EP300" "BRCA1"
## [4,] "GABPA"  "PPARGC1A" "MYOD1" "SMAD4" "SMAD4" "BCL6"
## [5,] "SP1"    "SMAD3"    "SMAD3" "SMAD3" "JDP2"  "EP300"
## [6,] "CREBBP" "ARNT"     "JUNB"  "JUND"  ""      "ESR1"
##      Topic_7 Topic_8  Topic_9 Topic_10
## [1,] "FOSL2" "EP300"  "EP300" "CREBBP"
## [2,] "JUND"  "GABPA"  "NCOA1" "PROX1"
## [3,] "EP300" "SP1"    "GTF2B" "MAPK1"
## [4,] "JUN"   "CREBBP" "ESRRA" "PRKCD"
## [5,] "SMAD3" "ATF1"   "ESR1"  "NCOA3"
## [6,] ""      ""       "NCOA3" "NCOA1"
```

The `igraph` networks are not created at the beginning, the `GenerateNetworks` should be used to convert the `topEdges` slot into networks.

```
> hlda <- GenerateNetworks(hlda)
> head(networks(hlda))
```

```
## $Network1
## IGRAPH c9f73ca UN-B 7 6 --
## + attr: name (v/c), type (v/c)
## + edges from c9f73ca (vertex names):
## [1] EP300 --MYOD1  EP300 --ARNT   MYOD1 --SP1
## [4] GABPA --SP1    MYOD1 --CREBBP CREBBP--SMAD3
##
## $Network2
## IGRAPH e649419 UN-B 6 9 --
## + attr: name (v/c), type (v/c)
```

```
## + edges from e649419 (vertex names):
## [1] ESR1 --ESRRA    ESR1 --NCOA1    ESRRA--NCOA1
## [4] ESR1 --PPARGC1A ESRRA--PPARGC1A NCOA1--PPARGC1A
## [7] ESR1 --SMAD3    ESR1 --ARNT     NCOA1--ARNT
##
## $Network3
## IGRAPH c5427a8 UN-B 6 5 --
## + attr: name (v/c), type (v/c)
## + edges from c5427a8 (vertex names):
## [1] FOSL2--JUND  FOSL2--JUN   JUN  --MYOD1 JUN  --SMAD3
## [5] FOSL2--JUNB
##
## $Network4
## IGRAPH 50783fd UN-B 11 15 --
## + attr: name (v/c), type (v/c)
## + edges from 50783fd (vertex names):
##  [1] GABPA--SP1   SP1  --MAPK1 SP1  --SMAD4 MAPK1--SMAD4
##  [5] MAPK1--SMAD3 MAPK1--JUND  SP1  --BCL6  MAPK1--BCL6
##  [9] SP1  --TBP   JUND --TBP   SP1  --MAPK3 JUND --MAPK3
## [13] SP1  --BRCA1 TBP  --ESR1  MAPK3--ESR1
##
## $Network5
## IGRAPH 6045f61 UN-B 5 5 --
## + attr: name (v/c), type (v/c)
## + edges from 6045f61 (vertex names):
## [1] ESR1 --JUND  ESR1 --EP300 ESR1 --SMAD4 EP300--SMAD4
## [5] EP300--JDP2
##
## $Network6
## IGRAPH 5164e65 UN-B 7 5 --
## + attr: name (v/c), type (v/c)
## + edges from 5164e65 (vertex names):
## [1] JUND --SMAD4 SMAD4--BRCA1 BCL6 --EP300 BRCA1--SMAD3
## [5] ESR1 --SMAD3
```

if the user wants to annotate each protein in the network by its gene expression he can use the `annotateExpression` method. To use this method the user needs to provide a `data.frame` object, that contains the names of the genes in the first column and their expression value in the second.

```
> data(RPKMS)
> hlda <- annotateExpression(hlda, RPKMS)
> networks(hlda)[[1]]

## IGRAPH c9f73ca UN-B 7 6 --
## + attr: name (v/c), type (v/c), RPKM (v/n)
## + edges from c9f73ca (vertex names):
## [1] EP300 --MYOD1  EP300 --ARNT   MYOD1 --SP1
## [4] GABPA --SP1    MYOD1 --CREBBP CREBBP--SMAD3
```

We can notice that the RPKM attribute was added to the network.

2.5 Cluster DNA interactions by enrichment profile

Till the moment we can say ok, we got the list of out networks and we can do further biological examination to check the results, however,it would be nice if we can know which DNA interactions are enriched for same networks. Thus, the package provide a clustering feature to further analysis.

The `clusterInteractions` can be used to do so. Two types of clustering can be done:

- Supervised : in which the user provides the number of clusters he wants, this part is done using the `sota` method of the *clValid* package.
- non-supervised : in which the number of clusters is determined automatically, performed using the `clues` method of the `clues` package.

The `clusterInteractions` is defined as follow:

```
clusterInteractions(object, method = c("clues", "sota"), nbClus = 20)
```

by default the `clues` method is chosen , if you specify `sota` the default number of clusters is 20.

```
> ## clustering using the 'clues' method
> hlda <- clusterInteractions(hlda, method = "clues")

## clusterInteractions : checking
## clusterInteractions : reading args
## using clues

## DNA interactions have been clustered into 7 cluster
```

2.6 Visualization

The package comes with a bunch of visualization plots to enable the exploration of data mainly through the `plot3CPETRes` method.

```
plot3CPETRes(object, path = "", W = 14, H = 7, type = c("heatmap",
    "clusters", "curve", "avgCurve", "netSim", "networks"), byEdge = TRUE,
    layoutfct = layout.kamada.kawai, ...)
```

To get the genomic coordinates of the regions involved in each cluster the methods `getRegionsIncluster` can be used. To use it the initial initial interaction data should be provided.

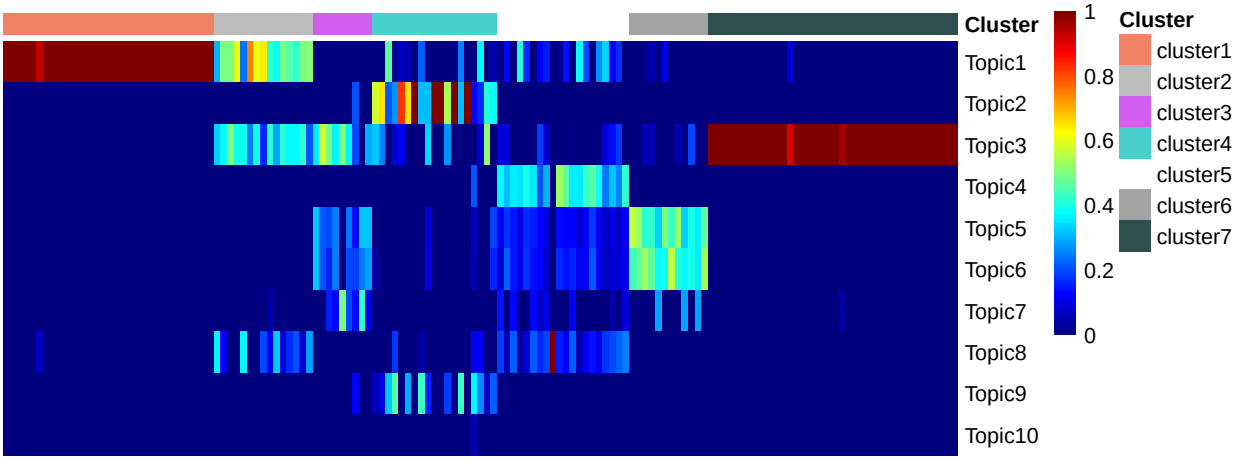
```
> getRegionsIncluster(hlda, x, cluster = 3)

## GRanges object with 18 ranges and 1 metadata column:
##         seqnames              ranges strand |      PET_ID
##            <Rle>           <IRanges>  <Rle> | <character>
##     [1]     chr1   27319369-27321369      * |    PET#16.1
##     [2]     chr1   27320398-27322398      * |    PET#16.2
##     [3]     chr1   93296540-93298540      * |    PET#44.1
##     [4]     chr1   93297761-93299761      * |    PET#44.2
##     [5]     chr2 216298272-216300272      * |   PET#181.1
##     ...      ...                 ...    ... .         ...
##    [14]     chr3 129374402-129376402      * |   PET#230.2
##    [15]     chr4     1794664-1796664      * |   PET#250.1
```

```
##   [16]     chr4     1796239-1798239      * |   PET#250.2
##   [17]     chr5 131130570-131132570      * |   PET#288.1
##   [18]     chr5 131131692-131133692      * |   PET#288.2
##   -------
##   seqinfo: 6 sequences from an unspecified genome; no seqlengths
```

2.6.1 Heatmaps

After clustering the enrichment map can be visualized using the `heatmap` option in the `plot3CPETRes` method. Here each column represent a chromatin-chromatin interaction and each row represents a chromatin maintainer network. The colors indicate the probability that a chromatin-chromatin interaction is maintained by a chromatin maintainer network.

```
> plot3CPETRes(hlda, type = "heatmap")
```

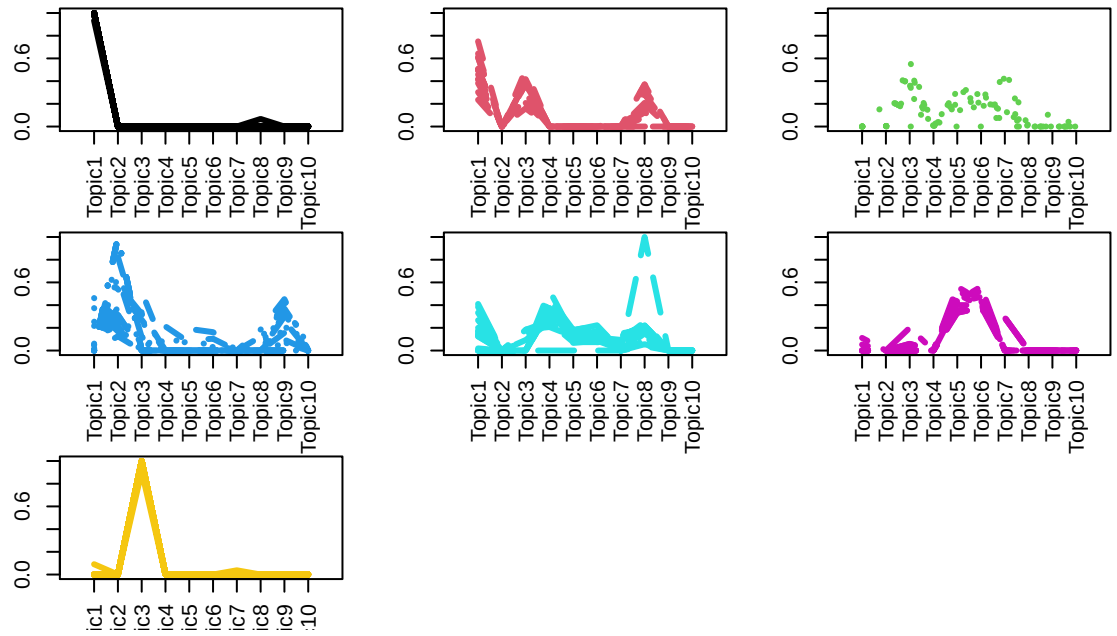


2.6.2 Enrichment curves

An other way to check the enrichment of the different chromatin interactions in the different clusters is by plotting the enrichment and average enrichment curves in of the chromatin interactions in each clusters.

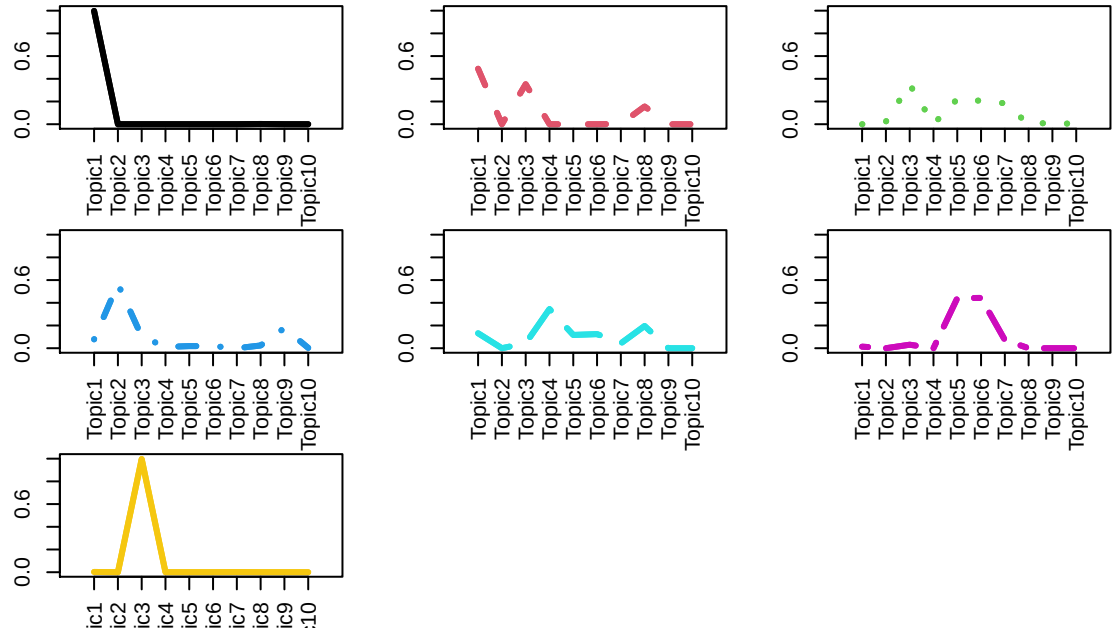
if the `type` parameter of the `plot3CPETRes` method is set to `curve` the enrichment profile of all the interactions per cluster is displayed as shown bellow:

```
> ## plotting curves
> plot3CPETRes(hlda, type = "curve")
```



if `type = "avgCurve"` then the average curves are displayed.

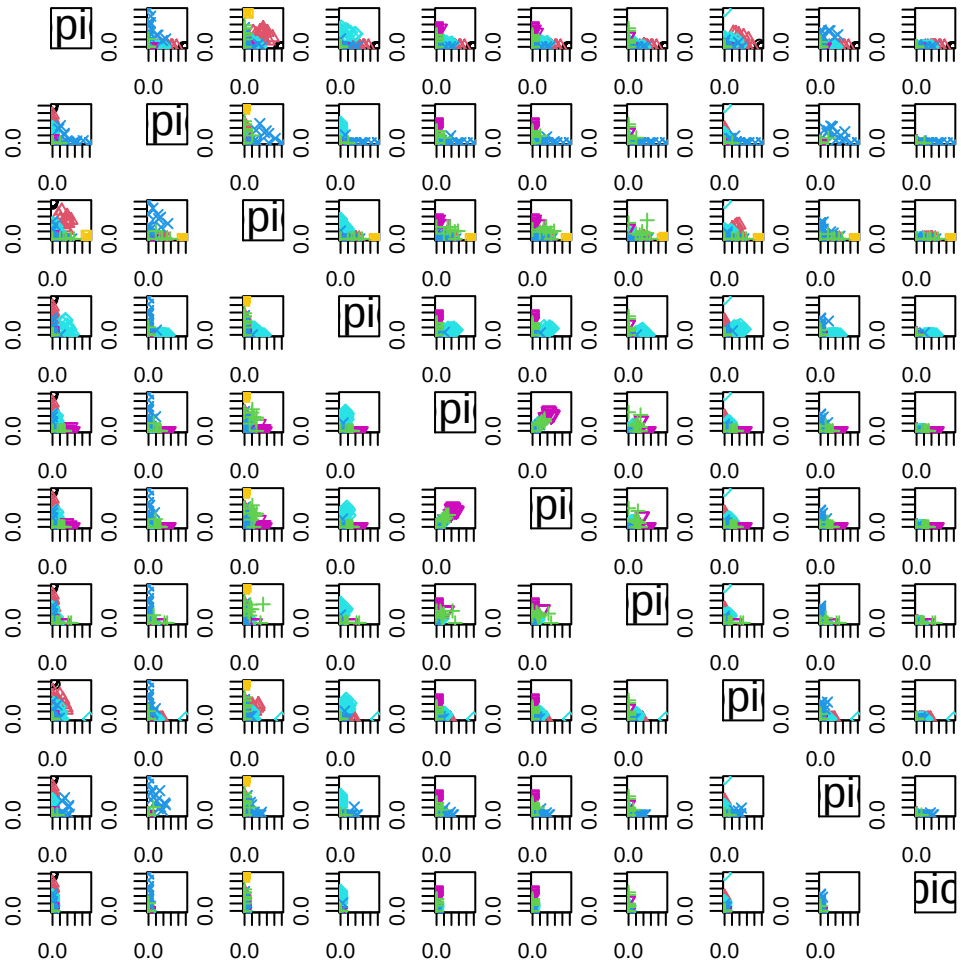
```
> ## plotting Average curves
> plot3CPETRes(hlda, type = "avgCurve")
```



2.6.3 clusters pair-wise scatter plot

In some cases we want to see which two clusters can give a better separation of the data, in this case the `type = "clusters"` option can be used. This option is only available if the clustering have been done using the `clues` method.

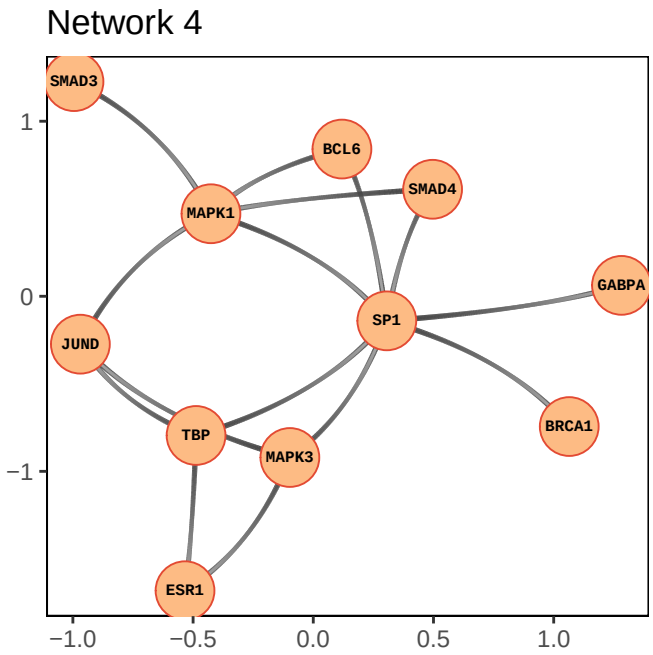
```
> ## plotting pair-wise clusters scatter plots
> plot3CPETRes(hlda, type = "clusters")
```



2.6.4 plot networks

if `type ="networks"` is used, a pdf file `AllGraphs.pdf` is created and contains one networks per page. This method returns a `ggplot` list (one for each network). By default the `layout.kamada.kawai` from the *igraph* package is used by the user can pass any other function through the `layoutfct` parameter.

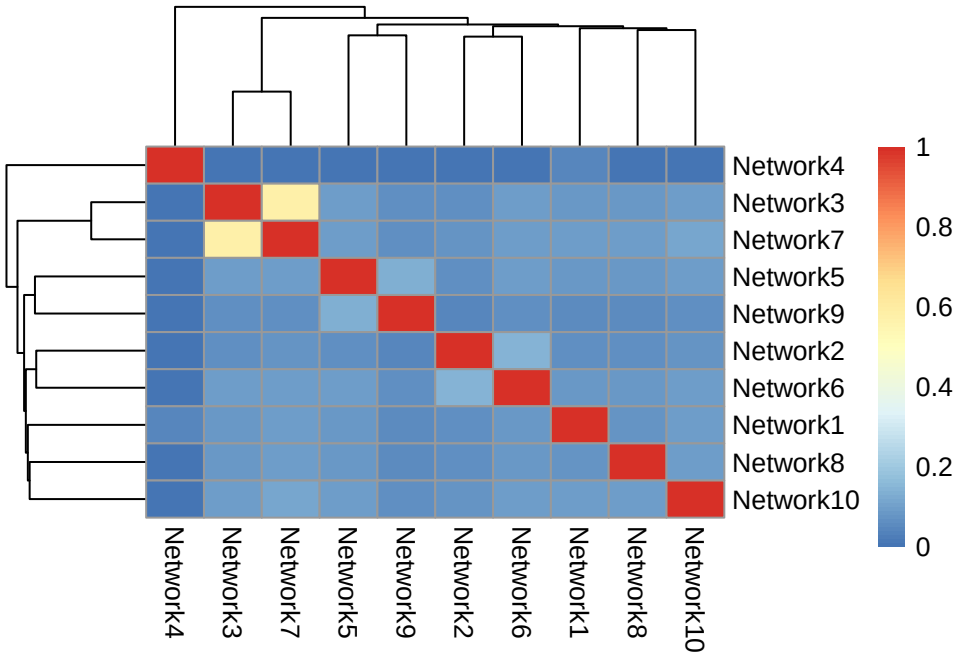
```
> nets_plot <- plot3CPETRes(hlda, type = "networks")
> plot(nets_plot[[4]])
```



2.6.5 Networks similarity

To the degree to which the infered networks share some common edges we can set `type = "netSim"`. of course, the smaller the similarity the better.

```
> plot3CPETRes(hlda, type = "netSim")
```



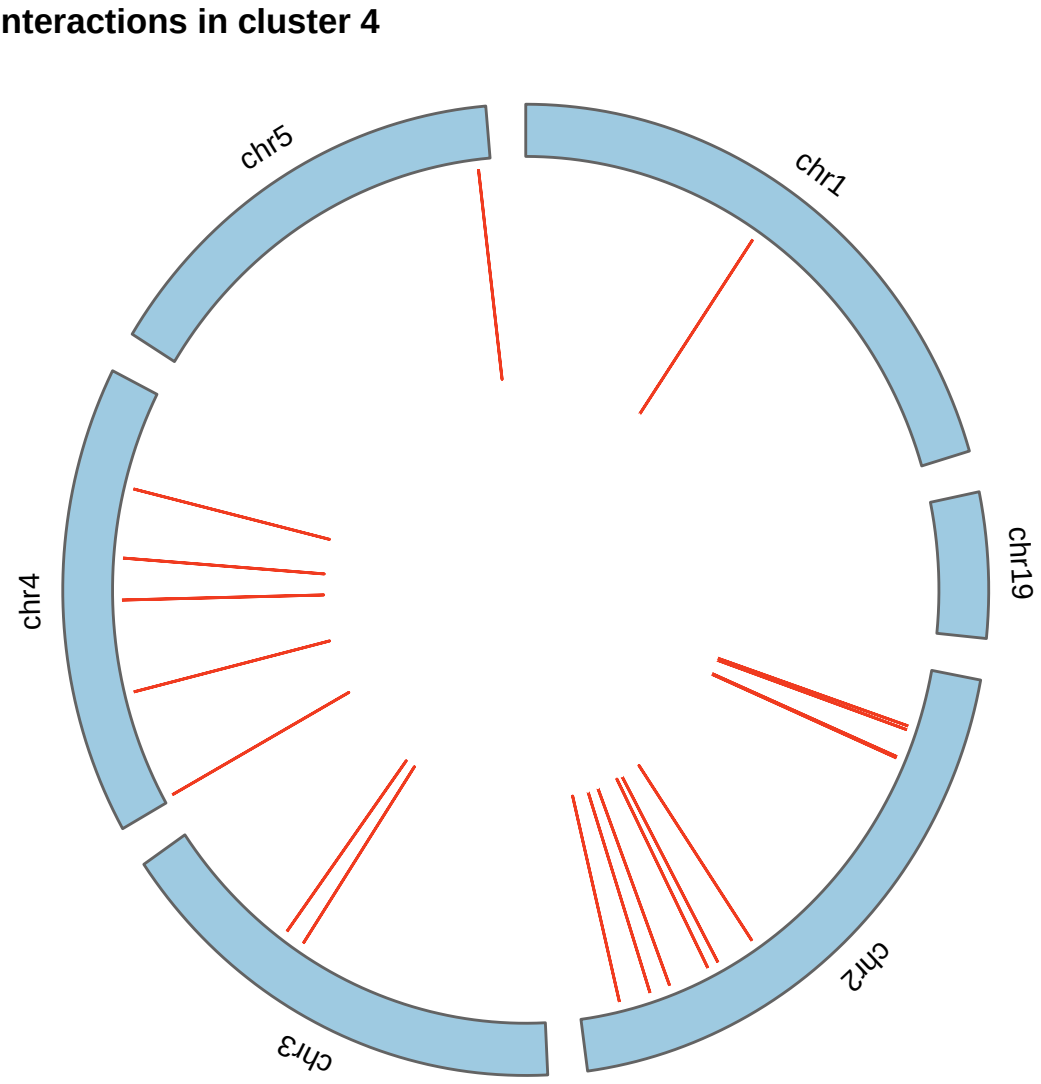
2.6.6 Circos maps

R3CPET also enable the user to plot a basic circos map for a given cluster through the method `visualizeCircos`. The initial data should be passed in the `data` parameter as a `ChiapetExperimentData` object.

```
visualizeCircos(object, data, cluster = 1, chrLenghts = NULL)
```

By default, the human chromosome lengths are used, if the user is using different species he can provide his own chromosome lengths as a two columns `data.frame` that contains the name of the chromosome in the first column and the length in the second one.

```
> visualizeCircos(hlda, x, cluster = 4)
```



2.7 Gene enrichment

R3CPET enable uses to do a GO analysis using the `DAVID` web service. Two types of enrichment can be done:

- GO enrichment analysis of the proteins of the inferred networks using the `GOEnrich.networks` method.

- GO enrichment of the chromatin interaction clusters using the `GOEnrich.folder` method.

2.7.1 Networks GO enrichment

```
> GOEnrich.networks(object, fdr = 0.05, GOlimit = 5, path = "")
```

The `GOEnrich.networks` method can be used for that. it takes as parameters a `ChromMaintainers` object and a optionally the cu-off `FDR` and the path to save the generated figure. The maximum number of returned GO terms per cluster can be specified by the `GOlimit` parameter.

```
> GOEnrich.networks(hlda, path = ".")
```

Each column in this plot represent a network and each row represents the GO term to which each network is enriched. The size of the dot is proportional to the number of percentage of the proteins enriched for that term.

2.7.2 GO enrichment of the genes in each cluster

Another question that we want to ask is what are the genes involved in each cluster ? and do they share some specific function. Similarly a GO annotation method is available. However, before doing the GO enrichment the list of list in each each cluster should be extracted, this can be done using the `outputGenesPerClusterToDir` method.

```
outputGenesPerClusterToDir(hdaRes, data, path = "ClustersGenes",
    ...)
```

This function generated a folder (by default named `ClustersGenes`) that contains `.txt` file for each cluster.

```
> outputGenesPerClusterToDir(hlda, x)
```

Then we can use the `GOEnrich.folder` to enrich the list of genes in the created folder.

```
> GOEnrich.folder(folder = "ClustersGenes/")
```

A figure will be generated for the significantly enriched gene lists (example figure bellow).

Figure 3: **Example of the GO enrichment for a genes of cluster 1**

2.8 Using the web interface

After getting all the results (HLDA, clustering, ... etc), the user can display the results using a web browser developped using the *shiny* package. This can be done throght the method `createServer`.

```
createServer(x, nets, hlda)
```

In the website the user can have some information about the raw data, such as the used TF, how many regeions each TF binds, the distribution of interaction per cluster, ... etc. Some of the features are explained in the flowing points

2.8.1 Raw data visualization

Statistics about the 3 types of raw data data (chromatin interactions, TFBS, PPI) can be displayed. Two select options are available under the "Raw Data" panel:

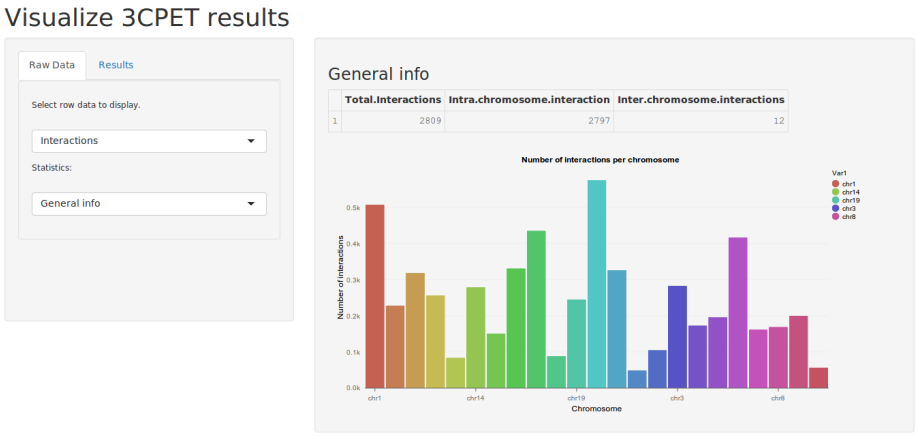


Figure 4: **Example showing the histogram of the number of interacting region per chromosome**

- data type selection option: in which the user selects the type of data he wants analyze : Interactions, TFBS or PPI.
- statistics selection option : in which the user selects the type of the plot he wants to generate (Figure 4)

2.8.2 Results visualization

This panel also enables the user to interactively analyze his results. Two types of results can be analyzed : The concerning the Chromatin maintainer networks, and the other one about the clustered genomic regions. For example, Figure 5. shows a screenshot in which the user selects a Chromatin maintainer network and display it in an interactive manner using the *D3js* javascript library.

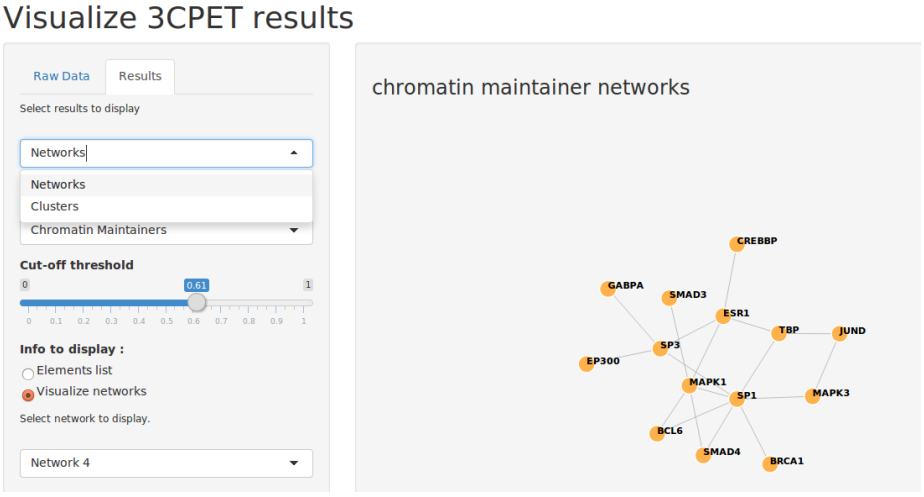


Figure 5: **Example of the plots in the web interface**

3 Session Info

```
sessionInfo()
```

```
## R version 4.0.0 alpha (2020-04-05 r78150)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices
## [6] utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_3.3.0        R3CPET_1.19.0
## [3] Rcpp_1.0.4.6         igraph_1.2.5
## [5] GenomicRanges_1.39.3 GenomeInfoDb_1.23.16
## [7] IRanges_2.21.8       S4Vectors_0.25.15
## [9] BiocGenerics_0.33.3
##
## loaded via a namespace (and not attached):
##   [1] colorspace_1.4-1
##   [2] ellipsis_0.3.0
##   [3] class_7.3-16
##   [4] biovizBase_1.35.1
##   [5] htmlTable_1.13.3
##   [6] XVector_0.27.2
##   [7] base64enc_0.1-3
##   [8] dichromat_2.0-0
##   [9] rstudioapi_0.11
##  [10] farver_2.0.3
##  [11] bit64_0.9-7
##  [12] AnnotationDbi_1.49.1
##  [13] fansi_0.4.1
##  [14] codetools_0.2-16
##  [15] splines_4.0.0
##  [16] ggbio_1.35.1
##  [17] knitr_1.28
##  [18] Formula_1.2-3
##  [19] Rsamtools_2.3.7
##  [20] cluster_2.1.0
##  [21] dbplyr_1.4.2
##  [22] png_0.1-7
##  [23] clues_0.6.2.2
##  [24] pheatmap_1.0.12
##  [25] graph_1.65.2
```

```
##  [26] BiocManager_1.30.10
##  [27] compiler_4.0.0
##  [28] httr_1.4.1
##  [29] backports_1.1.6
##  [30] assertthat_0.2.1
##  [31] Matrix_1.2-18
##  [32] lazyeval_0.2.2
##  [33] cli_2.0.2
##  [34] formatR_1.7
##  [35] acepack_1.4.1
##  [36] htmltools_0.4.0
##  [37] prettyunits_1.1.1
##  [38] tools_4.0.0
##  [39] gtable_0.3.0
##  [40] glue_1.4.0
##  [41] GenomeInfoDbData_1.2.2
##  [42] reshape2_1.4.3
##  [43] dplyr_0.8.5
##  [44] rappdirs_0.3.1
##  [45] Biobase_2.47.3
##  [46] vctrs_0.2.4
##  [47] Biostrings_2.55.7
##  [48] rtracklayer_1.47.0
##  [49] xfun_0.12
##  [50] stringr_1.4.0
##  [51] lifecycle_0.2.0
##  [52] ensembldb_2.11.3
##  [53] XML_3.99-0.3
##  [54] zlibbioc_1.33.1
##  [55] scales_1.1.0
##  [56] BiocStyle_2.15.6
##  [57] BSgenome_1.55.4
##  [58] VariantAnnotation_1.33.3
##  [59] hms_0.5.3
##  [60] ProtGenerics_1.19.3
##  [61] RBGL_1.63.1
##  [62] SummarizedExperiment_1.17.5
##  [63] AnnotationFilter_1.11.0
##  [64] RColorBrewer_1.1-2
##  [65] yaml_2.2.1
##  [66] curl_4.3
##  [67] memoise_1.1.0
##  [68] gridExtra_2.3
##  [69] biomaRt_2.43.5
##  [70] rpart_4.1-15
##  [71] reshape_0.8.8
##  [72] latticeExtra_0.6-29
##  [73] stringi_1.4.6
##  [74] RSQLite_2.2.0
##  [75] highr_0.8
##  [76] checkmate_2.0.0
```

```
##  [77] GenomicFeatures_1.39.7
##  [78] BiocParallel_1.21.2
##  [79] rlang_0.4.5
##  [80] pkgconfig_2.0.3
##  [81] matrixStats_0.56.0
##  [82] bitops_1.0-6
##  [83] evaluate_0.14
##  [84] lattice_0.20-41
##  [85] purrr_0.3.3
##  [86] labeling_0.3
##  [87] GenomicAlignments_1.23.2
##  [88] htmlwidgets_1.5.1
##  [89] bit_1.1-15.2
##  [90] tidyselect_1.0.0
##  [91] GGally_1.5.0
##  [92] plyr_1.8.6
##  [93] magrittr_1.5
##  [94] R6_2.4.1
##  [95] Hmisc_4.4-0
##  [96] DelayedArray_0.13.10
##  [97] DBI_1.1.0
##  [98] withr_2.1.2
##  [99] pillar_1.4.3
## [100] foreign_0.8-77
## [101] survival_3.1-11
## [102] RCurl_1.98-1.1
## [103] nnet_7.3-13
## [104] tibble_3.0.0
## [105] crayon_1.3.4
## [106] OrganismDbi_1.29.1
## [107] BiocFileCache_1.11.4
## [108] rmarkdown_2.1
## [109] jpeg_0.1-8.1
## [110] progress_1.2.2
## [111] grid_4.0.0
## [112] data.table_1.12.8
## [113] blob_1.2.1
## [114] digest_0.6.25
## [115] clValid_0.6-6
## [116] openssl_1.4.1
## [117] munsell_0.5.0
## [118] askpass_1.1
```