Package 'RBPGraph'

May 2, 2019

Type Package

Title Estimates RBP Co-Binding from eCLIP Data Using Partial Correlation Networks

Version 0.1.0

Author Jo-Jo Feng

Maintainer The package maintainer <jo-jo.feng@yale.edu>

Description This package provides functions to convert from eCLIP narrowPeak files to matrices that can be used to estimate partial correlation coefficients between RBPs to identify potential co-binding partners. Together, these functions compose a streamlined workflow for generating networks direclty from eCLIP peak files. We also provide a flexible and easy-to-use framework to view and analyze the graphs.

License

Encoding UTF-8

LazyData true

R topics documented:

`BedGraphsToBigWigs` *BedGraphsToBigWigs*

Description

Converts bedGraphs into bigWigs. Must write files to an output directory.

Usage

```
BedGraphsToBigWigs(inputDir, outputDir, chromSizes, bedGraphToBigWig = "bedGraphToBigWig")
```

Arguments

inputDir	Directory containing bedGraphs to convert to bigWigs.
outputDir	Output directory where bigWigs will be written.
chromSizes	Chromosome sizes file. Each row should be a chromosome name followed by its number of bases.
bedGraphToBigWig	Path to bedGraphToBigWig.

Details

This function makes a system call to bedGraphToBigWig for each bedGraph file.

Value

Returns nothing

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (inputDir, outputDir, chromSizes, bedGraphToBigWig = "bedGraphToBigWig")
{
    dir.create(outputDir, showWarnings = FALSE)
    for (file in list.files(path = inputDir, pattern = "*.bedGraph",
        full.names = TRUE, recursive = FALSE)) {
      outputFile <- paste0(file.path(outputDir, tools::file_path_sans_ext(basename(file))),
          ".bigWig")
        cmd <- paste(bedGraphToBigWig, file, chromSizes, outputFile)
        system(cmd)
    }
  }
```

BigWigsToMatrix *BigWigsToMatrix*

Description

For each bigWig file, averages scores across provided bins. Concatenates scores column-wise for each file into a single matrix.

Usage

```
BigWigsToMatrix(inputDir, binsFile, bigWigAverageOverBed = "bigWigAverageOverBed",
column = 5, outputDir = NULL)
```

Arguments

inputDir	The directory containing the bigWig files to average over bins.
binsFile	The file containing the bins (observations). Each row should contain a chromosome, start, stop and a unique name. This file must be sorted by 'sort -k1,1 -k2,2n'.
bigWigAverageOverBed	Path to bigWigAverageOverBed
column	Which column to use as the score for each bin. Column 3 takes the sum, 4 is the average over all bases of the bin, 5 is the average over just the covered bases. Defaults to 5 to avoid penalizing small peaks.
outputDir	Optional: Provide a directory if you would like to write the individual column files. If the directory does not exist, it will be created.

Details

This function will make a system call to bigWigAverageOverBed for each file in the inputDir. For best performance with bigWigAverageOverBed, increase available memory.

Value

BigWigsToMatrix returns a data.frame where each column corresponds to a bigWig file, and each row is a bin. The values will be sums or means depending on the bigWigAverageOverBed column selected.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (inputDir, binsFile, bigWigAverageOverBed = "bigWigAverageOverBed",
    column = 5, outputDir = NULL)
{
    if (!is.null(outputDir)) {
        dir.create(outputDir, showWarnings = FALSE)
    }
    else {
        outputDir <- tempdir()
    }
    mat <- NULL
    for (file in list.files(path = inputDir, pattern = "*.bigWig",
        full.names = TRUE, recursive = FALSE)) {
      outputFile <- paste0(file.path(outputDir, tools::file_path_sans_ext(basename(file))),
          "_averaged.bed")
        cmd <- paste(bigWigAverageOverBed, file, binsFile, outputFile)
        system(cmd)
        result <- data.table::fread(outputFile, data.table = FALSE)
        if (!is.null(outputDir)) {
          outputFile <- paste0(file.path(outputDir, tools::file_path_sans_ext(basename(file))),
              ".bed")
            write.table(result[, column], outputFile, sep = "\t",
                row.names = FALSE, col.names = FALSE, quote = FALSE)
        }
```

```
        if (is.null(mat)) {
            mat <- result[, column]
        }
        else {
            mat <- cbind(mat, result[, column])
        }
    }
    mat
}
```

---

eCLIPToBedGraph *eCLIPToBedGraph*

Description

Converts eCLIP narrowPeak files to bedGraphs.

Usage

```
eCLIPToBedGraph(inputDir, outputDir, column, qualityThreshold = 1000, splitStrands = TRUE)
```

Arguments

inputDir	Directory containing eCLIP narrowPeak files.
outputDir	Directory to which bedGraph files will be written.
column	The column number containing the score desired for analysis. Defaults to 7 (fold-enrichment).
qualityThreshold	Threshold for peak quality score to be included. Defaults to 1000 (i.e., all peaks with scores lower than 1000 will be ignored).
splitStrands	Boolean describing whether to split peaks from strands into two separate files. It is generally necessary to split strands to avoid overlapping peaks in the same file, which bedGraphToBigWig cannot handle. Splitting strands will require a call to MergeStrands on the final data matrix. Defaults to TRUE.

Details

Outputted bedGraphs are sorted according to 'sort -k1,1 -k2,2n'.

Value

Returns nothing

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (inputDir, outputDir, column, qualityThreshold = 1000,
    splitStrands = TRUE)
```

```
{
    for (file in list.files(path = inputDir, pattern = "*.bed",
        full.names = TRUE, recursive = FALSE)) {
        d <- data.table::fread(file, data.table = FALSE)
        if (splitStrands) {
          outputFile.plus <- paste0(file.path(outputDir, tools::file_path_sans_ext(basename(file))),
                "_plus.bedGraph")
          outputFile.minus <- paste0(file.path(outputDir, tools::file_path_sans_ext(basename(file))),
                "_minus.bedGraph")
            scores.plus <- d[d[, 5] >= qualityThreshold & d[,
                6] == "+", c(1, 2, 3, column)]
            scores.minus <- d[d[, 5] >= qualityThreshold & d[,
                6] == "-", c(1, 2, 3, column)]
            write.table(setorder(scores.plus), outputFile.plus,
                sep = "\t", row.names = FALSE, col.names = FALSE,
                quote = FALSE)
            write.table(setorder(scores.minus), outputFile.minus,
                sep = "\t", row.names = FALSE, col.names = FALSE,
                quote = FALSE)
        }
        else {
          outputFile <- paste0(file.path(outputDir, tools::file_path_sans_ext(basename(file))),
                ".bedGraph")
            scores <- d[d[, 5] >= qualityThreshold, c(1, 2, 3,
                column)]
            write.table(setorder(scores), outputFile, sep = "\t",
                row.names = FALSE, col.names = FALSE, quote = FALSE)
        }
    }
  }
```

`GenerateGraphs` *GenerateGraphs*

Description

Generates and plots estimated partial correlation networks.

Usage

```
GenerateGraphs(data, lambdas, rbps, approx)
```

Arguments

`data`	An m x n matrix representing m observations (bins) and n (RBPs).
`lambdas`	A vector of tuning parameters to use to regularize the generated networks. length(lambdas) total networks will be generated.
`rbps`	A vector containing names of RBPs (in the same order as the columns are ordered). length(rbps) must equal n.
`approx`	Boolean, whether to use the Meinshausen and Buhlmann method to approximate the covariance matrices. Defaults to FALSE.

Details

This function applies the nonparanormal transformation to the data before computing the correlation matrix, which it then feeds to the glassopath function. The resulting inverse covariance matrices are converted to graph weight matrices and plotted.

Value

A list with the components

w	Estimated covariance matrices, has dimensions (m,n, length(lambdas))
wi	Estimated inverse covariance matrix, has dimensions (m, n, length(lambdas))
approx	Input argument approx
lambdas	Vector of tuning parameters provided
errflag	Error value, 0 indicates no error
graphs	Weight matrices for each estimated graph, has dimensions (m, n, length(lambdas))

References

Epskamp, S., Cramer, A.O.J., Waldorp, L.J., Schmittmann, V.D., & Borsboom, D. (2012). qgraph: Network Visualizations of Relationships in Psychometric Data. Journal of Statistical Software, 48(4). http://dx.doi.org/10.18637/jss.v048.i04.

Friedman, J., Hastie, T., & Tibshirani, R. (2007). Sparse inverse covariance estimation with the graphical lasso. Biostatistics, 0(0). 1-10.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

set.seed(100)

x<-matrix(rnorm(50*3),ncol=3)
results <- GenerateGraphs(x, c(1e-3, 3e-3, 1e-2, 3e-2), c("RBP1", "RBP2", "RBP3"))

## The function is currently defined as
function(data, lambdas, rbps, approx=FALSE) {
# perform non-paranormal transformation, cannot assume RBP data is normally distributed
data.npn <- huge::huge.npn(data)

# perform correlation
data.cor <- qgraph::cor_auto(data.npn)

# get graphs for each of the given lambdas
results.path <- glasso::glassopath(data.cor, rholist=lambdas, trace=0, approx=approx)

# set up the layout
h <- floor(sqrt(length(lambdas)))
w <- ceiling(length(lambdas) / h)
layout(matrix(1:(h*w), h, w, byrow=TRUE))

# plot each graph,
results.path$graphs <- array(0.0, dim(results.path$wi))
```

```
for (i in 1:length(lambdas)) {
# convert estimated inverse-covariance matrix to weight matrix
results.path$graphs[,,i] <- as.matrix(qgraph::wi2net(results.path$wi[,,i]))
qgraph::qgraph(results.path$graphs[,,i], layout="spring", parallelEdge=TRUE, diag=FALSE,
directed=FALSE, theme="colorblind", cut=0, title=results.path$rholist[i], labels=rbps)
}
# rename for clarity
names(results.path)[4] <- "lambdas"
results.path
}
```

MergeStrands *MergeStrands*

Description

Merges adjacent columns of a matrix (assumed to be opposite strands of the same RBP) by combining the values either by sum or average.

Usage

```
MergeStrands(mat, fn = "mean")
```

Arguments

mat	Final data matrix, where each column corresponds to a strand of an RBP. If this function is being called, it is implied that eCLIPToBedGraph split strands; i.e., each eCLIP file yielded two bedGraphs.
fn	Describes how to combine the scores. Default is "mean", but can be changed to "sum".

Details

This function should be called on the final matrix if and only if splitStrands was set to TRUE in eCLIPToBedGraph. The input matrix should have 2n columns, where n is the total number of RBPs. As long as the file names created by upstream RBPGraph functions were left unchanged, opposing strands from the same RBP should be adjacent to one another, which is the assumption this function makes. Confirm that it is the case where files in the inputDir to bigWigsToMatrix were ordered such that each pair of adjacent files corresponds to the same RBP.

Value

Returns an m x n matrix, where the input was m x 2n. Adjacent pairs of columns are combined either by averaging the values or summing them.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

set.seed(100)
```

```
x<-matrix(rnorm(50*10),ncol=10)
x.merged <- MergeStrands(x)

## The function is currently defined as
function (mat, fn = "mean")
{
    divisor <- 2
    if (fn == "sum") {
        divisor <- 1
    }
    mat <- (mat[, seq(1, ncol(mat), by = 2)] + mat[, seq(2, ncol(mat),
        by = 2)])/divisor
    mat
  }
```

Index