Package 'BiBitR'

February 14, 2017

Type Package

Title R Wrapper for Java Implementation of BiBit

Version 0.2.2

Date 2017-02-10

Author De Troyer Ewoud

Maintainer De Troyer Ewoud <ewoud.detroyer@uhasselt.be>

Description A simple R wrapper for the Java BiBit algorithm from ``A biclustering algorithm for extracting bit-patterns from binary datasets'' from Domingo et al. (2011) <DOI:10.1093/bioinformatics/btr464>. An simple adaption for the BiBit algorithm which allows noise in the biclusters is also introduced. Further, a workflow to guide the algorithm towards given patterns is included as well.

License GPL-3

Imports foreign,methods,utils, biclust

RoxygenNote 5.0.1

SystemRequirements Java

R topics documented:

bibit *The BiBit Algorithm*

Description

A R-wrapper which directly calls the original Java code for the BiBit algorithm (http://eps.upo.es/bigs/BiBit.html) and transforms it to the output format of the Biclust R package.

Usage

```
bibit(matrix = NULL, minr = 2, minc = 2, arff_row_col = NULL,
  output_path = NULL)
```

Arguments

matrix	The binary input matrix.
minr	The minimum number of rows of the Biclusters.
minc	The minimum number of columns of the Biclusters.
arff_row_col	If you want to circumvent the internal R function to convert the matrix to .arff format, provide the pathname of this file here. Additionally, two .csv files should be provided containing 1 column of row and column names. These two files should not contain a header or quotes around the names, simply 1 column with the names. (*Example*: arff_row_col=c("...\\data\\matrix.arff","...\\data\\rownames.csv","...\\ *Note:* These files can be generated with the make_arff_row_col function.
output_path	If as output, the original txt output of the Java code is desired, provide the outputh path here (without extension). In this case the bibit function will skip the transformation to a Biclust class object and simply return NULL. (*Example*: output_path="...\\out\\bibitresult") (*Description Output*: The following information about every bicluster generated will be printed in the output file: number of rows, number of columns, name of rows and name of columns.

Details

This function uses the original Java code directly (with the intended input and output). Because the Java code was not refactored, the rJava package could not be used. The bibit function does the following:

1. Convert R matrix to a .arff output file.
2. Use the .arff file as input for the Java code which is called by system().
3. The outputted .txt file from the Java BiBit algorithm is read in and transformed to a Biclust object.

Because of this, there is a chance of *overhead* when applying the algorithm on large datasets. Make sure your machine has enough RAM available when applying to big data.

Value

A Biclust S4 Class object.

Author(s)

Ewoud De Troyer

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]
result <- bibit(data,minr=5,minc=5)
result
MaxBC(result)

## End(Not run)
```

---

bibit2 *The BiBit Algorithm with Noise Allowance*

Description

Same function as `bibit` with an additional new noise parameter which allows 0's in the discovered biclusters (See Details for more info).

Usage

```
bibit2(matrix = NULL, minr = 2, minc = 2, noise = 0,
  arff_row_col = NULL, output_path = NULL, extend_columns = FALSE)
```

Arguments

`matrix`	The binary input matrix.
`minr`	The minimum number of rows of the Biclusters.
`minc`	The minimum number of columns of the Biclusters.
`noise`	Noise parameter which determines the amount of zero's allowed in the bicluster (i.e. in the extra added rows to the starting row pair).

- `noise=0`: No noise allowed. This gives the same result as using the `bibit` function.
- `0<noise<1`: The `noise` parameter will be a noise percentage. The number of allowed 0's in a (extra) row in the bicluster will depend on the column size of the bicluster. More specifically `zeros_allowed = ceiling(noise * columnsize)`. For example for `noise=0.10` and a bicluster column size of 5, the number of allowed 0's would be 1.

- noise>=1: The noise parameter will be the number of allowed 0's in a (extra) row in the bicluster independent from the column size of the bicluster. In this noise option, the noise parameter should be an integer.

arff_row_col — If you want to circumvent the internal R function to convert the matrix to .arff format, provide the pathname of this file here. Additionally, two .csv files should be provided containing 1 column of row and column names. These two files should not contain a header or quotes around the names, simply 1 column with the names.
(*Example*: arff_row_col=c("...\\data\\matrix.arff","...\\data\\rownames.csv","...\\
Note: These files can be generated with the make_arff_row_col function.

output_path — If as output, the original txt output of the Java code is desired, provide the outputh path here (without extension). In this case the bibit function will skip the transformation to a Biclust class object and simply return NULL.
(*Example*: output_path="...\\out\\bibitresult")
(*Description Output*: The following information about every bicluster generated will be printed in the output file: number of rows, number of columns, name of rows and name of columns.

extend_columns — (EXPERIMENTAL!) Boolean value which applies a column extension procedure to the result of the BiBit algorithm. Columns will be sequentially added, keeping the noise beneath the allowed level. The procedure is the same as in bibit3, but now no artificial rows have to be ignored in the noise levels.
Note: The @info slot will also contain a BC.Extended value which contains the indices of which Biclusters's columns were extended.

Details

bibit2 follows the same steps as described in the Details section of bibit.
Following the general steps of the BiBit algorithm, the allowance for noise in the biclusters is inserted in the original algorithm as such:

1. Binary data is encoded in bit words.
2. Take a pair of rows as your starting point.
3. Find the maximal overlap of 1's between these two rows and save this as a pattern/motif. You now have a bicluster of 2 rows and N columns in which N is the number of 1's in the motif.
4. Check all remaining rows if they match this motif, *however* allow a specific amount of 0's in this matching as defined by the noise parameter. Those rows that match completely or those within the allowed noise range are added to bicluster.
5. Go back to *Step 2* and repeat for all possible row pairs.

Note: Biclusters are only saved if they satisfy the minr and minc parameter settings and if the bicluster is not already contained completely within another bicluster.

What you will end up with are biclusters not only consisting out of 1's, but biclusters in which 2 rows (the starting pair) are all 1's and in which the other rows could contain 0's (= noise).

Note: Because of the extra checks involved in the noise allowance, using noise might increase the computation time a little bit.

Value

A Biclust S4 Class object.

Author(s)

Ewoud De Troyer

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]

result1 <- bibit2(data,minr=5,minc=5,noise=0.2)
result1
MaxBC(result1,top=1)

result2 <- bibit2(data,minr=5,minc=5,noise=3)
result2
MaxBC(result2,top=2)

## End(Not run)
```

`bibit3` *The BiBit Algorithm with Noise Allowance guided by Provided Patterns.*

Description

Same function as `bibit2` but only aims to discover biclusters containing the (sub) pattern of provided patterns or their combinations.

Usage

```
bibit3(matrix = NULL, minr = 1, minc = 2, noise = 0,
  pattern_matrix = NULL, subpattern = TRUE, extend_columns = TRUE,
  pattern_combinations = FALSE, arff_row_col = NULL)
```

Arguments

`matrix`	The binary input matrix.
`minr`	The minimum number of rows of the Biclusters. (Note that in contrast to `bibit` and `bibit2`, this can be be set to 1 since we are looking for additional rows to the provided pattern.)
`minc`	The minimum number of columns of the Biclusters.
`noise`	Noise parameter which determines the amount of zero's allowed in the bicluster (i.e. in the extra added rows to the starting row pair).

- `noise=0`: No noise allowed. This gives the same result as using the `bibit` function.
- `0<noise<1`: The `noise` parameter will be a noise percentage. The number of allowed 0's in a (extra) row in the bicluster will depend on the column size of the bicluster. More specifically `zeros_allowed = ceiling(noise * columnsize)`. For example for `noise=0.10` and a bicluster column size of 5, the number of allowed 0's would be 1.
- `noise>=1`: The `noise` parameter will be the number of allowed 0's in a (extra) row in the bicluster independent from the column size of the bicluster. In this noise option, the noise parameter should be an integer.

`pattern_matrix` Matrix (Number of Patterns x Number of Data Columns) containing the patterns of interest.

`subpattern` Boolean value if sub patterns are of interest as well (default=TRUE).

`extend_columns` Boolean value if columns of Biclusters should also be extended for additional results (default=TRUE). See Details Section for more info.

`pattern_combinations`
Boolean value if the pairwise combinations of patterns (the intersecting 1's) should also used as starting points (default=FALSE).

`arff_row_col` Same argument as in `bibit` and `bibit2`. However you can only provide 1 pattern by using this option. For `bibit3` to work, the pattern has to be added 2 times on top of the matrix (= identical first 2 rows).

Details

The goal of the `bibit3` function is to provide one or multiple patterns in order to only find those biclusters exhibiting those patterns. Multiple patterns can be given in matrix format, `pattern_matrix`, and their pairwise combinations can automatically be added to this matrix by setting `pattern_combinations=TRUE`. All discovered biclusters are still subject to the provided `noise` level.

Three types of Biclusters can be discovered:

Full Pattern: Bicluster which overlaps completely (within allowed noise levels) with the provided pattern. The column size of this bicluster is always equal to the number of 1's in the pattern.

Sub Pattern: Biclusters which overlap with a part of the provided pattern within allowed noise levels. Will only be given if `subpattern=TRUE` (default). Setting this option to `FALSE` decreases computation time.

Extended: Using the resulting biclusters from the full and sub patterns, other columns will be attempted to be added to the biclusters while keeping the noise as low as possible (the number of rows in the BC stays constant). Naturally the articially added pattern rows will not be taken into account with the noise levels as they are 0 in each other column.
The question which is attempted to be answered here is *'Do the rows, which overlap partly or fully with the given pattern, have other similarities outside the given pattern¿*

How?
The BiBit algorithm is applied to a data matrix that contains 2 identical artificial rows at the top which contain the given pattern. The default algorithm is then slightly altered to only start from this articial row pair (=Full Pattern) or from 1 artificial row and 1 other row (=Sub Pattern).

Note 1 - Large Data:
The `arff_row_col` can still be provided in case of large data matrices, but the `.arff` file should already contain the pattern of interest in the first two rows. Consequently not more than 1 pattern at a time can be investigated with a single call of `bibit3`.

Note 2 - Viewing Results:
A `print` and `summary` method has been implemented for the output object of `bibit3`. It gives an overview of the amount of discovered biclusters and their dimensions
Additionally, the `bibit3_patternBC` function can extract a Bicluster and add the artificial pattern rows to investigate the results.

Value

A S3 list object, `"bibit3"` in which each element (apart from the last one) corresponds with a provided pattern or combination thereof.
Each element is a list containing:

`Number`: Number of Initially found BC's by applying BiBit with the provided pattern.

`Number_Extended`: Number of additional discovered BC's by extending the columns.

`FullPattern`: Biclust S4 Class Object containing the Bicluster with the Full Pattern.

`SubPattern`: Biclust S4 Class Object containing the Biclusters showing parts of the pattern.

`Extended`: Biclust S4 Class Object containing the additional Biclusters after extending the biclusters (column wise) of the full and sub patterns

`info`: Contains `Time_Min` element which includes the elapsed time of parts and the full analysis.

The last element in the list is a matrix containing all the investigated patterns.

Author(s)

Ewoud De Troyer

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

Examples

```
## Not run:
set.seed(1)
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
colsel <- sample(1:ncol(data),ncol(data))
data <- data[sample(1:nrow(data),nrow(data)),colsel]

pattern_matrix <- matrix(0,nrow=3,ncol=100)
pattern_matrix[1,1:7] <- 1
pattern_matrix[2,11:15] <- 1
pattern_matrix[3,13:20] <- 1

pattern_matrix <- pattern_matrix[,colsel]


out <- bibit3(matrix=data,minr=2,minc=2,noise=0.1,pattern_matrix=pattern_matrix,
              subpattern=TRUE,extend_columns=TRUE,pattern_combinations=TRUE)
out  # OR print(out) OR summary(out)
```

```
bibit3_patternBC(result=out,matrix=data,pattern=c(1),type=c("full","sub","ext"),BC=c(1,2))

## End(Not run)
```

`bibit3_patternBC` *Extract BC from* `bibit3` *result and add pattern*

Description

Function which will print the BC matrix and add 2 duplicate articial pattern rows on top. The function allows you to see the BC and the pattern the BC was guided towards to.

Usage

```
bibit3_patternBC(result, matrix, pattern = c(1), type = c("full", "sub",
  "ext"), BC = c(1))
```

Arguments

`result`	Result produced by `bibit3`
`matrix`	The binary input matrix.
`pattern`	Vector containing either the number or name of which patterns the BC results should be extracted.
`type`	Vector for which BC results should be printed. • Full Pattern (`"full"`) • Sub Pattern (`"sub"`) • Extended (`"ext"`)
`BC`	Vector of BC indices which should be printed, conditioned on `pattern` and `type`.

Value

Prints queried biclusters.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
set.seed(1)
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
colsel <- sample(1:ncol(data),ncol(data))
data <- data[sample(1:nrow(data),nrow(data)),colsel]

pattern_matrix <- matrix(0,nrow=3,ncol=100)
```

```
pattern_matrix[1,1:7] <- 1
pattern_matrix[2,11:15] <- 1
pattern_matrix[3,13:20] <- 1

pattern_matrix <- pattern_matrix[,colsel]


out <- bibit3(matrix=data,minr=2,minc=2,noise=0.1,pattern_matrix=pattern_matrix,
             subpattern=TRUE,extend_columns=TRUE,pattern_combinations=TRUE)
out  # OR print(out) OR summary(out)


bibit3_patternBC(result=out,matrix=data,pattern=c(1),type=c("full","sub","ext"),BC=c(1,2))

## End(Not run)
```

`BiBitR` *A biclustering algorithm for extracting bit-patterns from binary datasets*

Description

BiBit R is a simple R wrapper which directly calls the original Java code for applying the BiBit algorithm. The original Java code can be found at http://eps.upo.es/bigs/BiBit.html by Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz.

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

`GOF` *Computing Fitness Score of Biclustering Result*

Description

EXPERIMENTAL FUNCTION, still needs tuning. Will eventually be integrated in bibit2 function.

Usage

```
GOF(matrix, bicresult, alpha = 1, verbose = FALSE)
```

Arguments

`matrix`	The binary input matrix.
`bicresult`	A `Biclust` result. (e.g. The return object from `bibit` or `bibit2`)
`alpha`	Weighting factor between 0 and 1.
`verbose`	Boolean value to show a short summary.

Value

A list containing the scores.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]

result1 <- bibit2(data,minr=5,minc=5,noise=0.2)
result1

fitness <- GOF(matrix=data,bicresult=result1,alpha=0.5)
summary(fitness)

## End(Not run)
```

---

make_arff_row_col *Transform R matrix object to BiBit input files.*

Description

Transform the R matrix object to 1 `.arff` for the data and 2 `.csv` files for the row and column names. These are the 3 files required for the original BiBit Java algorithm The path of these 3 files can then be used in the `arff_row_col` parameter of the `bibit` function.

Usage

```
make_arff_row_col(matrix, name = "data", path = "")
```

Arguments

`matrix`	The binary input matrix.
`name`	Basename for the 3 input files.
`path`	Directory path where to write the 3 input files to.

Value

3 input files for BiBit:

- 1 `.arff` file containing the data.
- 1 `.csv` file for the row names. The file contains 1 column of names without quotation.
- 1 `.csv` file for the column names. The file contains 1 column of names without quotation.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]

make_arff_row_col(matrix=data,name="data",path="")

result <- bibit(data,minr=5,minc=5,
               arff_row_col=c("data_arff.arff","data_rownames.csv","data_colnames.csv"))

## End(Not run)
```

MaxBC *Finding Maximum Size Biclusters*

Description

Simple function which scans a `Biclust` result and returns which biclusters have maximum row, column or size (row*column).

Usage

```
MaxBC(bicresult, top = 1)
```

Arguments

`bicresult`	A `Biclust` result. (e.g. The return object from `bibit` or `bibit2`)
`top`	The number of top row/col/size dimension which are searched for. (e.g. default `top=1` gives only the maximum)

Value

A list containing:

- `$row`: A matrix containing in the columns the Biclusters which had maximum rows, and in the rows the Row Dimension, Column Dimension and Size.
- `$column`: A matrix containing in the columns the Biclusters which had maximum columns, and in the rows the Row Dimension, Column Dimension and Size.
- `$size`: A matrix containing in the columns the Biclusters which had maximum size, and in the rows the Row Dimension, Column Dimension and Size.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]
result <- bibit(data,minr=2,minc=2)

MaxBC(result)


## End(Not run)
```

---

rows_full1_in_BC *Finding BC's with specific rows which only 1's in the BC.*

Description

Simple function which scans a `Biclust` result and returns which biclusters contain all rows given in the `rows` parameter, but only if these rows only contain 1's in the bicluster. This can be particularly helpful after having added articial row-pairs with a pattern of interest. With this function you can retrieve the biclusters that grew from these pairs from all the discovered biclusters.

Usage

```
rows_full1_in_BC(matrix, bicresult, rows)
```

Arguments

`matrix`	The binary input matrix.
`bicresult`	A `Biclust` result. (e.g. The return object from `bibit` or `bibit2`)
`rows`	A vector containing containing the row numbers which should be in the bicluster.

Value

A matrix containing the biclusters in the columns and the row, column and size dimensions on the rows.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
```

```
extra_rows <- rep(0,100)
extra_rows[11:25] <- 1

data <- rbind(data,rbind(extra_rows,extra_rows))
rownames(data) <- NULL

result <- bibit2(data,minr=2,minc=2,noise=0.2)

rows_full1_in_BC(matrix=data,bicresult=result,rows=c(101,102))


## End(Not run)
```

---

rows_in_BC *Finding BC's with specific rows.*

Description

Simple function which scans a Biclust result and returns which biclusters contain all rows given in the rows parameter.

Usage

```
rows_in_BC(bicresult, rows)
```

Arguments

bicresult	A Biclust result. (e.g. The return object from bibit or bibit2)
rows	A vector containing containing the row numbers which should be in the bicluster.

Value

A matrix containing the biclusters in the columns and the row, column and size dimensions on the rows.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
result <- bibit(data,minr=2,minc=2)

rows_in_BC(result,rows=c(21,22,23))


## End(Not run)
```

Index