Package ‘cloneid’

April 22, 2021

Type Package

Title Model-Based Systems for tracing and steering clonal dynamics

Version 1.1.0

Date 2017-03-03

Author Noemi Andor

Maintainer Noemi Andor `<cloneid.r@gmail.com>`

Description CLONEID has a three-tier architecture: a SQL database in the back-end, a Java core and an R user interface. The database consists of tables associated with experimental and computational aspects of in-vitro studies. They come together to form two modules: the lineage tracing module keeps track of the pedigree of lineages grown in a lab and monitors changes in their phenotypes, such as growth rate, via computer vision. The multi-omics module links subclonal omics profiles from different high throughput assays to each other and to the phenotypes from the 1st module. CLONEID captures information for learning genotype-phenotype associations, but with an emphasis on monitoring phenotype over longer periods of time (i.e. phenotypic information has an additional temporal dimension). Perspective and Identity are SQL tables central to CLONEID's -omics module. They hold clone-specific profiles quantified with a given assay. For example, a clone's genome perspective may be inferred from exome-sequencing, while its transcriptome perspective may rely on single cell RNA sequencing. Entries in the SQL table Identity are clones that have been confirmed by at least two different assays (either same assay ran on biological replicates or different types of assays ran on the same cell sample).

License GPL-2

URL `https://github.com/noemiandor/cloneid`

Depends R (>= 3.5.0), rJava (>= 0.5-0)

Imports raster, RMySQL, qualV, gplots(>= 3.0.1), gdata (>= 2.17.0), RColorBrewer, gtools, flexclust, Matrix, expands, liayson (>= 1.0.0), matlab (>= 1.0.2), ape

NeedsCompilation no

Repository CRAN

Date/Publication 2017-04-03 08:11:04

RoxygenNote 6.0.1

R topics documented:

`CloneProfiles` *Data formatting example for clonal decompositions*

Description

Clonal profiles called from scRNA sequencing data of the SNU-16 stomach cancer cell line. Data structure describes input format for function @viewPerspective().

Usage

```
data(CloneProfiles)
```

Format

List with keys denoting file names and values denoting file contents expected by @viewPerspective().

Clonal profiles are given at two layers of cellular resolution: first layer contains subpopulations of cells (i.e. clones); second layer contains members (e.g. cells) assigned to each layer 1 subpopulation. Both layers are represented in the list. Entries are as follows:

SNU-16.spstats - layer 1 data.frame with each row denoting a subpopulation and columns denoting their IDs and cellular fractions

SNU-16.sps.cbs - layer 1 data.frame with each row denoting a genome feature (e.g. gene or copy number segment) and first column "LOCUS" denoting the name of that feature. Each additional column denotes the name of a subpopulation. Subpopulation names must include their respective cellular fraction as listed in SNU-16.spstats.

SNU-16.0.0978009.sps.cbs - optional. Layer 2 data.frame mapped to the name of the respective layer 1 subpopulation. Each row denotes a genome feature and first column "LOCUS" denotes the name of that feature. Each additional column denotes the name of a member from the respective layer 1 subpopulation.

SNU-16.0.1066227.sps.cbs - optional. same as above, but for a different subpopulation

SNU-16.0.1212664.sps.cbs - optional. same as above, but for a different subpopulation

SNU-16.0.1243724.sps.cbs - optional. same as above, but for a different subpopulation

SNU-16.0.1506865.sps.cbs - optional. same as above, but for a different subpopulation

SNU-16.0.1914997.sps.cbs - optional. same as above, but for a different subpopulation

SNU-16.0.2122133.sps.cbs - optional. same as above, but for a different subpopulation.

Examples

```
## Memory needed to save single-cell data into SQL database
options(java.parameters = "-Xmx7g")

load('~/Downloads/CloneProfiles.rda');
names(CloneProfiles)
sapply(CloneProfiles, dim)
print(CloneProfiles$`SNU-16.spstats`)

## Layer 1:
head(CloneProfiles[['SNU-16.sps.cbs']])

## Layer 2:
head(CloneProfiles[['SNU-16.0.1914997.sps.cbs']])[,1:5]

## Compare rows and columns between layer1 and layer2:
i2 = grep("SNU-16.0.", names(CloneProfiles), value=T)
sapply(CloneProfiles[i2], function(layer2) intersect(layer2$LOCUS, CloneProfiles[['SNU-16
for (clone in CloneProfiles$`SNU-16.spstats`$`Mean Weighted`){
  f = substr(as.character(clone), 1,8)
  print(paste("Profile for clone", clone, "is in column", grep(f, colnames(CloneProfiles$
  print(paste("Members of clone", clone, "are in entry", grep(f, names(CloneProfiles), va
}

## Save input for viewPerspective()
# setwd("~/Downloads")
# for (x in names(CloneProfiles)){
#   write.table(CloneProfiles[[x]], file = x, sep = "\t", quote = F, row.names = F)
# }

## Run viewPerspective() to add to SQL database:
# viewPerspective(pathToSample="~/Downloads/SNU-16.spstats", clonesDIR="~/Downloads", suf
```

`createCloneidSchema` *Create CLONEID Schema in MySQL*

Description

Function to create the CLONEID Schema in MySQL defined by the connection configuration yaml set in editCloneidConfig()

Usage

```
createCloneidSchema(forceCreateSchema = FALSE)
```

Arguments

`forceCreateSchema`	(boolean) If TRUE the current CLONEID database will be dropped and the schema recreated. WARNING: This will delete all data in CLONEID

Examples

```
# Create the CLONEID MySQL Schema (safe)
createCloneidSchema(forceCreateSchema = FALSE)

# Create the CLONEID MySQL Schema AND drop CLONEID database if exists
createCloneidSchema(forceCreateSchema = TRUE)
```

`editCloneidConfig` *Edit CloneID Configuration Yaml*

Description

Function to take user input to set CloneID's MySQL connection: host, port, user, password, database and schema build script location

Usage

```
editCloneidConfig(host = 'localhost', port = '3306', user = NA, password = NA, d
```

Arguments

`host`	string MySQL host domain url
`port`	string MySQL port number, default: 3306
`user`	string MySQL user to connect with
`password`	string MySQL user password, this is used by other CloneID functions to read/write to the MySQL database
`database`	string MySQL database name, default: CLONEID
`schemaScript`	string path to SQL script to build the CLONEID database

Examples

```
# To see current yaml configuration values:
editCloneidConfig()

# To set all values:
editCloneidConfig('localhost', '3306', 'user1', 'password1', 'CLONEID', 'CLONEID_schema.s

# To set select values:
editCloneidConfig(host='localhost')

editCloneidConfig(port='3306', user='user1')
```

`feed` *Database update: record feeding.*

Description

Database update function for table Passaging.

Usage

```
 feed(id, tx=Sys.time())
```

Arguments

`id`	The ID of the seeding event of the cells that are being fed. This will be used to search the Passaging table.
`tx`	Timestamp at which the seeding was performed. Defaults to current system time.

Details

Entry associated with the seeding ID will be updated with a timestamp indicating a feeding.

Author(s)

Noemi Andor

`findAllDescendandsOf` *Search database for all descendants of a lineage*

Description

Database search function for table Passaging.

Usage

```
findAllDescendandsOf(ids, mydb = NULL, recursive=T)
```

Arguments

`ids`	Character vector with each entry holding an ID of a lineage (i.e. a key in the Passaging table).
`mydb`	Object used to communicate with the database engine. If set to NULL, a new object is created.
`recursive`	Whether to return the immediate descendants of the IDs or all progeny.

Details

For each input ID, returns all lineages from the Passaging table that can be traced back to this ID.

Author(s)

Noemi Andor

`getPedigreeTree` *Phylogeny depicting how cell line lineages are related.*

Description

Visualizes relations between lineages of a cell line.

Usage

```
getPedigreeTree(cellLine = cellLine, id = NULL)
```

Arguments

`cellLine`	Name of the cell line for which to plot the lineage tree. If set to NULL, next argument will be used.
`id`	ID of the lineage for which to plot the tree of all derived sublineages. This must be an existing key in the Passaging table.

Author(s)

Noemi Andor

`getState` *Retrieving state associated with a clone*

Description

The state in which a clone was sequenced

Usage

```
getState(cloneID, whichP="TranscriptomePerspective"
```

Arguments

`cloneID`	Clone ID (integer).
`whichP`	What to vizualize: GenomePerspective (default), TranscriptomePerspective or Identity.

Value

A string describing subclones' state.

Author(s)

Noemi Andor

`getSubclones` *Retrieving subclones*

Description

Given the name of a biosample or the ID of a clone, the method retrieves all its subclones.

Usage

```
getSubclones(cloneID_or_sampleName,whichP="GenomePerspective")
```

Arguments

`cloneID_or_sampleName`	Clone ID (integer) or biosample name (character).
`whichP`	What to vizualize: GenomePerspective (default), TranscriptomePerspective or Identity.

Value

A map of each clone to its unique ID.

Author(s)

Noemi Andor

`getSubProfiles` *Retrieving subclone profiles*

Description

Given the name of a biosample or the ID of a clone, the method retrieve the profiles of all its subclones.

Usage

```
getSubProfiles(cloneID_or_sampleName,whichP="TranscriptomePerspective", includeR
```

Arguments

`cloneID_or_sampleName`	Clone ID (integer) or biosample name (character).
`whichP`	What to vizualize: GenomePerspective (default), TranscriptomePerspective or Identity.
`includeRoot`	Whether or not to include the parent clone's profile into the output.

Value

A matrix with rows corresponding to features and columns corresponding to subclones.

Author(s)

Noemi Andor

Examples

```
  pm=getSubProfiles(cloneID_or_sampleName = "LGG2T1",whichP = "GenomePerspective", incluc
```

`harvest` *New database entry: at any time before harvesting cells from a flask.*

Description

Database update function for table Passaging.

Usage

```
harvest(id, from, cellCount, tx = Sys.time(), media=NULL)
```

Arguments

`id`	The prefix of the file holding brightfield image(s) associated with this harvest event. This will be used as key in the Passaging table.
`from`	The ID of the seeding event from which these cells were harvested.
`cellCount`	Technician's best guess on how many cells are in the flask at the time of harvest. This is only for comparison with cell count inferred from segmentation algorithm and will not be saved to the database.
`tx`	Timestamp at which the harvest was performed. Defaults to current system time.
`media`	ID of the media used to grow these cells (key in SQL table 'Media').

Details

CLONEID's lineage tracing module streamlines routine monitoring of three aspects of in-vitro experiments: (i) the pedigree of all cell lineages grown in a lab; (ii) the exact media composition in which the cells grow and (iii) how often cells divide. Automatic recording of this information in the SQL table passaging requires technicians to adopt two new habits: (1) taking brightfield images of live cells both, immediately after seeding and any number of times before harvest. (2) Passing the images to CLONEID's seed and harvest functions respectively. This function facilitates the latter.

Author(s)

Noemi Andor

`hyper` *Subpopulation relatedness assessment*

Description

Calculates probability that two clones are related, modelling overlapping mutations as hypergeometric distribution.

Usage

```
hyper(p,r=NULL)
```

Arguments

`p`	A numeric vector or matrix holding the mutation profile of one or multiple clones (0 denotes absence; values >0 denote presence).
`r`	The mutation profile of the other clone. Can be NULL if p is a matrix.

Details

Let SP_P be a clone within one perspective and SP_R , a clone within another perspective of the same tumor (perspectives may be of same type). Further, let M_P, M_R be the set of SNVs assigned to SP_P and SP_R, while M_{PR} is the set of overlapping SNVs between SP_P and SP_R. We calculate how likely is it to observe at least | M_{PR} | common SNVs between SP_P and SP_R just by chance, by calculating: **a**) how likely it is to observe at least | M_{PR} | common SNVs in SP_P **b**) how likely it is to observe at least | M_{PR} | common SNVs in SP_R Both probabilities are modeled as Hypergeometric distributions: For (**b**), we draw x $\in M_R$ and each time x $\in$ MP we count the draw as success. Conversely, if x $\notin M_P$, the draw is considered a failure. For (**a**) we proceed in the same way, but reverse the role of SP_R with that of SP_P. The likelihood that SP_P is related to SP_R is calculated as the minimum among the two probabilities (**a,b**).

Value

Probability that SP_P is related to SP_R.

Author(s)

Noemi Andor

`merge` *Merging perspectives into clones' identities*

Description

Merges either two different perspectives on the clonal composition of the same specimen OR the same perspective on the clonal composition of two or more different specimens to approximate each clone's identity as the consensus across perspectives/specimens. Resulting consensus profiles are added to the SQL table 'Identity'. No new entry is made if no two perspectives agree on the existence of any clone.

Usage

```
merge(perspectives, specimens, simM = "euclidean", t=-Inf)
```

Arguments

`perspectives`	Minimal one and maximal two perspectives on a specimen. Two perspectives are required if only one specimen is provided.
`specimens`	The name of the specimen(s). Exactly one specimen is required if more than one perspectives are provided. Two or more specimens are required if only one perspective is provided.
`simM`	What similarity measure to use in order to match clonal components across perspectives/specimens. Options are: inverse of euclidean distance (euclidean - default), correlation coefficient (either pearson or spearman), mutation overlap significance as assessed by hypergeometric distribution (hyper).
`t`	Minimum similarity threshold below which two subpopulations will no longer be merged.

Details

Let S_i be the set of subpopulations detected in sample i and $S := \bigcup S_i$ – the set of clones detected across all biopsies of a given patient. Further let L be the set of all non-private loci, in which an SNV is detected in at least two samples and $M_x \subset L$ the set of loci mutated in $x \in S_i$.

Next, subpopulations S are grouped into categories by hierarchical cluster analysis of their SNV profiles M_S, using a distance metric defined by the hypergeometric probability calculated above (agglomeration method: "single"). Subpopulations from distinct samples, falling within the same category (hypergeometric $P \geq t$) are considered different perspectives on the same clone.

Value

List with four fields:

`sp2clone`	Matrix with rows denoting clones and columns holding the different perspectives on a clone. Entries contain the size of each clone and its ID in the database. Last column contains the Identity of each clone calculated across the preceding columns.
`sp2clone_sim`	Matrix with rows denoting clones and columns holding the different perspectives on a clone. Entries contain a measure of how confidently a clone could be assigned to the clone from the preceding column.
`consdat`	The consensus profile of each clone.
`usedOrder`	The path taken through persepctives to match clones.

Author(s)

Noemi Andor

Examples

```
#par(mfrow=c(4,1))
#display(cloneID_or_sampleName = "KATOIII",whichP = "GenomePerspective")display(cloneID
#display(cloneID_or_sampleName,whichP="TranscriptomePerspective", colorBy = NULL, deep
#merge(perspectives=c("GenomePerspective", "TranscriptomePerspective"), specimens="KATO
##compare(4,1,perspective1 = "GenomePerspective",perspective2 = "Identity")
```

`seed` *New database entry: Seeding cells to a new flask for growth.*

Description

Database update function for table Passaging.

Usage

```
seed(id, from, cellCount, dishSurfaceArea_cm2, tx = Sys.time(), media=NULL)
```

Arguments

`id`	The prefix of the file holding brightfield image(s) associated with this seeding event. This will be used as key in the Passaging table.
`from`	The ID of the harvest event from which these cells were seeded.
`cellCount`	Technician's best guess on how many cells are in the flask at the time of seeding. This is only for comparison with cell count inferred from segmentation algorithm and will not be saved to the database.
`dishSurfaceArea_cm2`	The size of the flask in which the cells were seeded.
`tx`	Timestamp at which the seeding was performed. Defaults to current system time
`media`	ID of the media used to grow these cells (key in SQL table 'Media').

Details

CLONEID's lineage tracing module streamlines routine monitoring of three aspects of in-vitro experiments: (i) the pedigree of all cell lineages grown in a lab; (ii) the exact media composition in which the cells grow and (iii) how often cells divide. Automatic recording of this information in the SQL table passaging requires technicians to adopt two new habits: (1) taking brightfield images of live cells both, immediately after seeding and any number of times before harvest. (2) Passing the images to CLONEID's seed and harvest functions respectively. This function facilitates the former.

Author(s)

Noemi Andor

`viewPerspective` *Biosample's subclonal composition*

Description

Reads the subclonal composition of a biosample and adds it to the SQL table 'Perspective'.

Usage

```
viewPerspective(spstatsFile, whichP, suffix = ".sps.cbs", xy = NULL)
```

Arguments

`spstatsFile`	The path to an .spstats file: data.frame with each row denoting a subpopulation and columns denoting their IDs and cellular fractions. The the file must be named according to the sample of origin of the profiled cells. The file must be in a directory containing the output of a clonal decomposition algorithm. See `CloneProfiles` for format requirements of output files.
`whichP`	What this assay provides: GenomePerspective or TranscriptomePerspective.
`suffix`	The suffix of the file within the output directory, containing the desired perspective.
`xy`	Two-dimensional vector containing the geographic location of the specimen.

Details

Profiles of the subpopulations listed in the .spstats file will be added to the SQL table 'Perspective'.

Author(s)

Noemi Andor

Index