Package ‘flowMatch’

May 28, 2024

Type Package

Title Matching and meta-clustering in flow cytometry

Version 1.40.0

Date 2015-03-26

Author Ariful Azad

Maintainer Ariful Azad <azad@lbl.gov>

Description Matching cell populations and building meta-clusters and templates from a collection of FC samples.

License Artistic-2.0

LazyLoad yes

Depends R (>= 3.0.0), Rcpp (>= 0.11.0), methods, flowCore

LinkingTo Rcpp

Imports Biobase

Suggests healthyFlowData

RcppModules flowMatch_module

biocViews ImmunoOncology, Clustering, FlowCytometry

git_url https://git.bioconductor.org/packages/flowMatch

git_branch RELEASE_3_19

git_last_commit e80de10

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-28

Contents

`flowMatch-package` *Matching cell populations and building meta-clusters and templates from a collection of FC samples.*

Description

Matching cell populations and building meta-clusters and templates from a collection of FC samples.

Details

Package:	flowMatch
Type:	Package
Version:	1.0
Date:	2013-08-01
License:	GPL (>= 2)
LazyLoad:	yes

Author(s)

Ariful Azad <aazad@purdue.edu>

References

Azad, Ariful and Pyne, Saumyadipta and Pothen, Alex (2012), Matching phosphorylation response patterns of antigen-receptor-stimulated T cells via flow cytometry; BMC Bioinformatics, 13 (Suppl 2), S10.

Azad, Ariful and Langguth, Johannes and Fang, Youhan and Qi, Alan and Pothen, Alex (2010), Identifying rare cell populations in comparative flow cytometry; Algorithms in Bioinformatics, Springer, 162-175.

Examples

```
## ------------------------------------------------
## load data
## ------------------------------------------------

library(healthyFlowData)
data(hd)

## ------------------------------------------------
## Retrieve each sample, clsuter it and store the
## clustered samples in a list
## ------------------------------------------------
set.seed(1234) # for reproducable clustering
cat('Clustering samples: ')
clustSamples = list()
for(i in 1:length(hd.flowSet))
{
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
}

## ------------------------------------------------
## Create a template from the list of clustered samples and plot functions
## ------------------------------------------------

template = create.template(clustSamples)
summary(template)

## plot the tree denoting the hierarchy of the samples in a template
tree = template.tree(template)

## plot the template in terms of the meta-clusters
## option-1 (default): plot contours of each cluster of the meta-clusters
plot(template)
```

`Cluster-class` *Cluster: A class representing a cell population in FC*

Description

An object of class "`Cluster`" represents a cluster or a cell population. We model a cluster with a normal distribution. An object of class "`Cluster`" therefore represents a cluster with a mean vector, a covariance matrix and the size of the cluster.

Creating Object

An object of class `Cluster` is usually created when constructing an object of class `ClusteredSample`. Unless you know exactly what you are doing, creating an object of class "`Cluster`" using `new` or using the constructor is discouraged.

An object of class "`Cluster`" can be created using the following constructor

```
Cluster(size, center, cov, cluster.id = NA_integer_, sample.id=NA_integer_)
```

The arguments of the constructor bear usual meaning as described in the value section above.

Slots

An object of class "`Cluster`" contains the following slots:

An integer denoting the number of points (cells) present in the cluster.

`size`: `center`: A numeric vector denoting the center of the cluster.

`cov`: A matrix denoting the covariances of the underlying normal distribution of the cluster.

`cluster.id`: The index of the cluster (relative to other clusters in same sample). Default is `NA_integer_` .

`sample.id`: The index of sample in which the cluster belongs to. Default is `NA_integer_` .

Accessors

All the slot accessor functions take an object of class `Cluster`. I show usage of the first accessor function. Other functions can be called similarly.

Returns the number of cells in the cluster.

Usage: `get.size(object)`

here `object` is a `Cluster` object.

`get.size`: `get.center`: Returns the center of the cluster.

`get.cov`: Returns the covariances matrix of the cluster.

`get.cluster.id`: Returns the index of the cluster (relative to other clusters in same sample).

`get.sample.id`: Returns the index of sample in which the cluster belongs to.

`sample.id<-`: Set the index of sample in which the cluster belongs to.

Methods

show Display details about the `Cluster` object.

summary Return descriptive summary for each `Cluster` object.

Usage: `summary(Cluster)`

Author(s)

Ariful Azad

See Also

ClusteredSample

Examples

```
## An object of class "Cluster"" is usually created when constructing a "ClusteredSample".
## Unless you know exactly what you are doing, creating an object of class "Cluster"
## using new or using the constructor is discouraged.

## ------------------------------------------------
## load data and retrieve a sample
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample = exprs(hd.flowSet[[1]])

## ------------------------------------------------
## cluster sample using kmeans algorithm
## and retrive the parameters of  the first cluster
## ------------------------------------------------

km = kmeans(sample, centers=4, nstart=20)
center1 = km$centers[1,]
# compute the covariance matrix of the first cluster
cov1 = cov(sample[km$cluster==1,])
size1 = length(which(km$cluster==1))

## ------------------------------------------------
## Create an object of class "Cluster"
## and show summary
## ------------------------------------------------

clust = Cluster(size=size1, center=center1, cov=cov1)
summary(clust)
```

ClusteredSample-class *ClusteredSample: A class representing a clustered FC Sample*

Description

An object of class "ClusteredSample" represents a partitioning of a sample into clusters. We model a flow cytometry sample with a mixture of cell populations where a cell population is a normally distributed cluster. An object of class "ClusteredSample" therefore stores a list of clusters and other necessary parameters.

Creating Object

An object of class "ClusteredSample" can be created using the following constructor

```
ClusteredSample(labels, centers=list(), covs=list(), sample=NULL, sample.id=NA_integer_)
```

- `labels` A vector of integers (from `1:num.clusters`) indicating the cluster to which each point is allocated. This is usually obtained from a clustering algorithm.
- `centers` A list of length `num.clusters` storing the centers of the clusters. The ith entry of the list `centers[[i]]` stores the center of the ith cluster. If not specified, the constructor estimates `centers` from `sample`.
- `covs` A list of length `num.clusters` storing the covariance matrices of the clusters. The ith entry of the list `cov[[i]]` stores the covariance matrix of the ith cluster. If not specified, the constructor estimates `cov` from `sample`.
- `sample` A matrix, data frame of observations, or object of class `flowFrame`. Rows correspond to observations and columns correspond to variables. It must be passed to the constructor if either `centers` or `cov` is unspecified; then `centers` or `cov` is estimated from `sample`.
- `sample.id` The index of the sample (relative to other samples of a cohort).

Slots

An object of class "ClusteredSample" contains the following slots:

The number of clusters in the sample.

`num.clusters` `labels` A vector of integers (from range `1:num.clusters`) indicating the cluster to which each point is assigned to. For example, `labels[i]=j` means that the ith element (cell) is assigned to the jth cluster.

`dimension` Dimensionality of the sample (number of columns in data matrix).

`clusters` A list of length `num.clusters` storing the cell populations. Each cluster is stored as an object of class `Cluster`.

`size` Number of cells in the sample (summation of all cluster sizes).

`sample.id` integer, denoting the index of the sample (relative to other samples of a cohort). Default is `NA_integer_`

Accessors

All the slot accessor functions take an object of class `ClusteredSample`. I show usage of the first accessor function. Other functions can be called similarly.

Returns the number of cells in the sample (summation of all cluster sizes).
Usage: `get.size(object)`
here `object` is a `ClusteredSample` object.

`get.size` `get.num.clusters` Returns the number of clusters in the sample.

`get.labels` Returns the cluster labels for each cell. For example, `labels[i]=j` means that the ith element (cell) is assigned to the jth cluster.

`get.dimension` Returns the dimensionality of the sample (number of columns in data matrix).

`get.clusters` Returns the list of clusters in this sample. Each cluster is stored as an object of class `Cluster`.

`get.sample.id` Returns the index of the sample (relative to other samples of a cohort).

Methods

show Display details about the `ClusteredSample` object.

summary Return descriptive summary for the `ClusteredSample` object.

Usage: `summary(ClusteredSample)`

plot We plot a sample by bivariate scatter plots where different clusters are shown in different colors.

Usage:

```
plot(sample, ClusteredSample, ...)
```

the arguments of the plot function are:

- `sample:` A matrix, data.frame or an object of class `flowFrame` representing an FC sample.
- `ClusteredSample:` An object of class `ClusteredSample` storing the clustering of the sample.
- `...` Other usual plotting related parameters.

Author(s)

Ariful Azad

See Also

`Cluster`

Examples

```
## ------------------------------------------------
## load data and retrieve a sample
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample = exprs(hd.flowSet[[1]])

## ------------------------------------------------
## cluster sample using kmeans algorithm
## ------------------------------------------------
km = kmeans(sample, centers=4, nstart=20)
cluster.labels = km$cluster

## ------------------------------------------------
## Create ClusteredSample object  (Option 1 )
## without specifying centers and covs
## we need to pass FC sample for paramter estimation
## ------------------------------------------------

clustSample = ClusteredSample(labels=cluster.labels, sample=sample)

## ------------------------------------------------
```

```
## Create ClusteredSample object  (Option 2)
## specifying centers and covs
## no need to pass the sample
## ------------------------------------------------

centers = list()
covs = list()
num.clusters = nrow(km$centers)
for(i in 1:num.clusters)
{
  centers[[i]] = km$centers[i,]
  covs[[i]] = cov(sample[cluster.labels==i,])
}
# Now we do not need to pass sample
ClusteredSample(labels=cluster.labels, centers=centers, covs=covs)

## ------------------------------------------------
## Show summary and plot a clustered sample
## ------------------------------------------------

summary(clustSample)
plot(sample, clustSample)
```

`ClusterMatch-class` *ClusterMatch: A class representing matching of cluster/meta-clusters across a pair of FC samples/templates*

Description

An object of class "`ClusterMatch`" represents matching of cluster/meta-clusters across a pair of FC samples/templates. A cluster (meta-cluster) from a sample (template) can match to zero, one or more than one cluster (meta-clusters) in another sample (template).

Creating Object

An object of class "`ClusterMatch`" is usually created by calling the function `match.clusters`:

`match.clusters(object1, object2, dist.type='Mahalanobis', unmatch.penalty=999999)`.

Here, `object1` and `object2` are two objects of class `ClusteredSample` or `Template` between which the clusters or meta-clusters are matched. See the example section and also the `match.clusters` function for more details.

Unless you know exactly what you are doing, creating an object of class "`ClusterMtach`" using `new` or using the constructor is discouraged.

Slots

Let S1 and S2 be two FC samples or templates with k1 and k2 clusters or meta-clusters respectively. Then the matching of clusters (meta-clusters) across S1 and S2 is represented by an object of class "ClusterMatch" that contains the following slots:

A list of length k1 where match12[[i]] stores the indices of clusters (meta-clusters) from S2 matched to the i-th clustrer (meta-cluster) of S1. match12[[i]]=NULL if the i-th cluster (meta-cluster) of S1 remains unmatched, otherwise, it stores a vector of matched clusters (meta-clusters) from S2.

match21: A list of length k2 where match21[[i]] stores the indices of clusters (meta-clusters) from S1 matched to the i-th clustrer (meta-cluster) of S2. match21[[i]]=NULL if the i-th cluster (meta-cluster) of S2 remains unmatched, otherwise, it stores a vector of matched clusters (meta-clusters) from S1.

matching.cost: The cost of matching clusters (meta-clusters) across the samples. It is equal to the summation of dissimilarities of the matched clusters (meta-clusters) and penalty for the unmatched clusters (meta-clusters).

unmatch.penalty: A numeric value denoting the penalty for leaving a cluster (meta-cluster) unmatched. If we set it to a a very large value then no cluster (meta-cluster) remains unmatched giving an edge cover solution.

Accessors

All the slot accessor functions take an object of class ClusterMatch. I show usage of the first accessor function. Other functions can be called similarly.

Returns the matching from cluster in sample 1 to clusters in sample 2. See the slot description for details. *Usage:* get.match12(object)

here object is a ClusterMatch object.

get.match21: Returns the matching from cluster in sample 2 to clusters in sample 1. See the slot description for details.

get.matching.cost: Returns the total cost of matching clusters (meta-clusters) across the pair samples/templates.

get.unmatch.penalty: Returns the penalty for leaving a cluster (meta-cluster) unmatched.

Methods

show Display details about the ClusterMatch object.

summary Return descriptive summary of the matching of clusters (meta-clusters) across a pair of samples (templates). Shows both list and matrix format.

Usage: summary(ClusterMatch)

Author(s)

Ariful Azad

See Also

match.clusters, ClusteredSample, Template

Examples

```
## ------------------------------------------------
## load data and retrieve two samples
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample1 = exprs(hd.flowSet[[1]])
sample2 = exprs(hd.flowSet[[2]])

## ------------------------------------------------
## cluster sample using kmeans algorithm
## ------------------------------------------------

clust1 = kmeans(sample1, centers=4, nstart=20)
clust2 = kmeans(sample2, centers=4, nstart=20)
cluster.labels1 = clust1$cluster
cluster.labels2 = clust2$cluster

## ------------------------------------------------
## Create ClusteredSample object
## and compute mahalanobis distance between two clsuters
## ------------------------------------------------

clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
clustSample2 = ClusteredSample(labels=cluster.labels2, sample=sample2)
## compute the dissimilarity matrix
DM = dist.matrix(clustSample1, clustSample2, dist.type='Mahalanobis')

## ------------------------------------------------
## Computing matching of clusteres
## An object of class "ClusterMatch" is returned
## ------------------------------------------------

mec = match.clusters(clustSample1, clustSample2, dist.type="Mahalanobis", unmatch.penalty=99999)
## show the matching
summary(mec)


## ************************************************************************
## ************** Now matching meta-clusters across templates ***********
## ************************************************************************

## ------------------------------------------------
## Retrieve each sample, clsuter it and store the
## clustered samples in a list
## ------------------------------------------------

cat('Clustering samples: ')
clustSamples = list()
for(i in 1:10) # read 10 samples and cluster them
{
```

```
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
}

## ------------------------------------------------
## Create two templates each from five samples
## ------------------------------------------------

template1 = create.template(clustSamples[1:5])
template2 = create.template(clustSamples[6:10])

## ------------------------------------------------
## Match meta-clusters across templates
## ------------------------------------------------

mec = match.clusters(template1, template2, dist.type="Mahalanobis", unmatch.penalty=99999)
summary(mec)

## ------------------------------------------------
## Another example of matching meta-clusters & clusters
## across a template and a sample
## ------------------------------------------------

mec = match.clusters(template1, clustSample1, dist.type="Mahalanobis", unmatch.penalty=99999)
summary(mec)
```

`create.template` *Creating a template of a collection of FC samples*

Description

Create an object of class `Template` summarizes a group of samples belonging to same biological-class with a class-template. A template is represented by a collection of meta-clusters (`MetaCluster`) created from samples of same class. An object of class `Template` therefore stores a list of `MetaCluster` objects and other necessary parameters.

Usage

```
create.template(clustSamples, dist.type = "Mahalanobis", unmatch.penalty=999999, template.id = NA_in
```

Arguments

`clustSamples`	A list of `ClusteredSample` objects from which the template is created. The working examples describe how this objects are created by clustering FC samples.

`dist.type` character, indicating the method with which the dissimilarity between a pair of clusters is computed. Supported dissimilarity measures are: 'Mahalanobis', 'KL' and 'Euclidean'. If this argument is not passed then 'Mahalanobis' distance is used by default.

`unmatch.penalty` A numeric value denoting the penalty for leaving a cluster unmatched. This parameter should be already known or be estimated empirically estimated from data (see the reference for a discussion). Default is set to a very high value so that no cluster remains unmatched.

`template.id` integer, denoting the index of the template (relative to other template). Default is NA_integer_

Details

An object of class `Template` summarizes a group of samples belonging to same biological-class with a class-specific template. A template is represented by a collection of meta-clusters (`MetaCluster`) created from samples of same class. An object of class `Template` therefore stores a list of `MetaCluster` objects and other necessary parameters.

Value

`dist.sample` returns a numeric value representing dissimilarity between a pair of samples. This value is equal to the summation of dissimilarities of the matched clusters and penalty for the unmatched clusters.

Author(s)

Ariful Azad

References

Azad, Ariful and Pyne, Saumyadipta and Pothen, Alex (2012), Matching phosphorylation response patterns of antigen-receptor-stimulated T cells via flow cytometry; BMC Bioinformatics, 13 (Suppl 2), S10.

See Also

`Template`, `MetaCluster`

Examples

```
 ## ------------------------------------------------
## load data
## ------------------------------------------------

library(healthyFlowData)
data(hd)

## ------------------------------------------------
## Retrieve each sample, clsuter it and store the
```

```
## clustered samples in a list
## ------------------------------------------------
set.seed(1234) # for reproducable clustering
cat('Clustering samples: ')
clustSamples = list()
for(i in 1:length(hd.flowSet))
{
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
}

## ------------------------------------------------
## Create a template from the list of clustered samples and plot functions
## ------------------------------------------------

template = create.template(clustSamples)
summary(template)

## plot the tree denoting the hierarchy of the samples in a template
tree = template.tree(template)


## plot the template in terms of the meta-clusters
## option-1 (default): plot contours of each cluster of the meta-clusters
plot(template)

## option-2: plot contours of each cluster of the meta-clusters with defined color
plot(template, color.mc=c('blue','black','green3','red'))

## option-3: plot contours of the meta-clusters with defined color
plot(template, plot.mc=TRUE, color.mc=c('blue','black','green3','red'))


## option-4: plot contours of each cluster of the meta-clusters with different colors for different samples
plot(template, colorbysample=TRUE)
```

`dist.cluster` *Dissimilarity between a pair of clusters*

Description

Calculate the dissimilarity between a pair of cell populations (clusters) from the distributions of the clusters.

Usage

```
dist.cluster(cluster1,cluster2, dist.type = 'Mahalanobis')
```

Arguments

`cluster1`	an object of class `Cluster` representing the distribution parameters of the first cluster.
`cluster2`	an object of class `Cluster` representing the distribution parameters of the second cluster.
`dist.type`	character, indicating the method with which the dissimilarity between a pair of clusters is computed. Supported dissimilarity measures are: 'Mahalanobis', 'KL' and 'Euclidean'.

Details

Consider two p-dimensional, normally distributed clusters with centers $\mu 1, \mu 2$ and covariance matrices $\Sigma 1, \Sigma 2$. Assume the size of the clusters are `n1` and `n2` respectively. We compute the dissimilarity `d12` between the clusters as follows:

1. If dist.type='Mahalanobis': we compute the dissimilarity `d12` with the Mahalanobis distance between the distributions of the clusters.

$$\Sigma = ((n1-1)*\Sigma 1 + (n2-1)*\Sigma 2)/(n1+n2-2)$$

$$d12 = sqrt(t(\mu 1 - \mu 2)*\Sigma^{(} - 1)*(\mu 1 - \mu 2))$$

2. If dist.type='KL': we compute the dissimilarity `d12` with the Symmetrized Kullback-Leibler divergence between the distributions of the clusters. Note that KL-divergence is not symmetric in its original form. We converted it symmetric by averaging both way KL divergence. The symmetrized KL-divergence is not a metric because it does not satisfy triangle inequality.

$$d12 = 1/4*(t(\mu 2-\mu 1)*(\Sigma 1^{(}-1)+\Sigma 2^{(}-1))*(\mu 2-\mu 1)+trace(\Sigma 1/\Sigma 2+\Sigma 2/\Sigma 1)+2p)$$

3. If dist.type='Euclidean': we compute the dissimilarity `d12` with the Euclidean distance between the centers of the clusters.

$$d12 = sqrt(\sum(\mu 1 - \mu 2)^2)$$

The dimension of the clusters must be same.

Value

`dist.cluster` returns a numeric value denoting the dissimilarities between a pair of cell populations (clusters).

Author(s)

Ariful Azad

References

McLachlan, GJ (1999) Mahalanobis distance; Journal of Resonance 4(6), 20–26.

Abou–Moustafa, Karim T and De La Torre, Fernando and Ferrie, Frank P (2010) Designing a Metric for the Difference between Gaussian Densities; Brain, Body and Machine, 57–70.

See Also

mahalanobis.dist, symmetric.KL, dist.matrix

Examples

```
## ------------------------------------------------
## load data and retrieve a sample
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample = exprs(hd.flowSet[[1]])

## ------------------------------------------------
## cluster sample using kmeans algorithm
## ------------------------------------------------

km = kmeans(sample, centers=4, nstart=20)
cluster.labels = km$cluster

## ------------------------------------------------
## Create ClusteredSample object
## and compute mahalanobis distance between two clsuters
## ------------------------------------------------

clustSample = ClusteredSample(labels=cluster.labels, sample=sample)
clust1 = get.clusters(clustSample)[[1]]
clust2 = get.clusters(clustSample)[[2]]
dist.cluster(clust1, clust2, dist.type='Mahalanobis')
dist.cluster(clust1, clust2, dist.type='KL')
dist.cluster(clust1, clust2, dist.type='Euclidean')
```

---

dist.matrix — *Dissimilarity matrix between each pair of clusters/meta-clusters across a pair of samples/templates*

Description

Calculate a matrix storing the dissimilarities between each pair of clusters (meta-clusters) across a pair of samples (templates) S1 and S2. (i,j)th entry of the matrix stores dissimilarity between i-th cluster (meta-cluster) from S1 and the j-th cluster (meta-cluster) from S2.

Usage

```
dist.matrix(object1,object2, dist.type = 'Mahalanobis')
```

Arguments

object1	an object of class ClusteredSample or Template.
object2	an object of class ClusteredSample or Template.
dist.type	character, indicating the method with which the dissimilarity between a pair of clusters (meta-clusters) is computed. Supported dissimilarity measures are: 'Mahalanobis', 'KL' and 'Euclidean', with the default is set to 'Mahalanobis' distance.

Details

Consider two FC samples/templates S1 and S2 with k1 and k2 clusters/meta-clusters. The dissimilarity between each pair of cluster (meta-clusters) across S1 and S2 is computed and stored in a (k1 x k2) matrix. The dissimilarity between i-th cluster (meta-cluster) from S1 and j-th cluster (meta-cluster) from S2 is computed using function dist.cluster.

Value

dist.matrix function returns a (k1 x k2) matrix where k1 and k2 are the number of clusters (meta-clusters) in the first and the second samples (templates) respectively. (i,j)th entry of the matrix contains the dissimilarity between the i-th cluster (meta-cluster) from sample1 (template1) and the j-th cluster (meta-cluster) from sample2 (template2).

Author(s)

Ariful Azad

See Also

dist.cluster

Examples

```
## ------------------------------------------------
## load data and retrieve two samples
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample1 = exprs(hd.flowSet[[1]])
sample2 = exprs(hd.flowSet[[2]])

## ------------------------------------------------
## cluster sample using kmeans algorithm
## ------------------------------------------------

clust1 = kmeans(sample1, centers=4, nstart=20)
```

```
clust2 = kmeans(sample2, centers=4, nstart=20)
cluster.labels1 = clust1$cluster
cluster.labels2 = clust2$cluster

## ------------------------------------------------
## Create ClusteredSample object
## and compute the Mahalanobis distance between
## each pair of clsuters and save it in a matrix
## ------------------------------------------------

clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
clustSample2 = ClusteredSample(labels=cluster.labels2, sample=sample2)
## compute the dissimilarity matrix
DM = dist.matrix(clustSample1, clustSample2, dist.type='Mahalanobis')
print(DM)
```

`dist.sample` *Dissimilarity between a pair of clustered FC samples*

Description

Compute the dissimilarity between a pair of clustered FC samples by using Mixed Edge Cover (MEC) algorithm.

Usage

```
dist.sample(clustSample1, clustSample2, dist.type='Mahalanobis', unmatch.penalty=999999)
```

Arguments

`clustSample1`	an object of class `ClusteredSample` containing cell populations from sample 1.
`clustSample2`	an object of class `ClusteredSample` containing cell populations from sample 2.
`dist.type`	character, indicating the method with which the dissimilarity between a pair of clusters is computed. Supported dissimilarity measures are: 'Mahalanobis', 'KL' and 'Euclidean'.
`unmatch.penalty`	A numeric value denoting the penalty for leaving a cluster unmatched. This parameter should be already known or be estimated empirically estimated from data (see the reference for a discussion). Default is set to a very high value so that no cluster remains unmatched.

Details

We used a robust version of matching called Mixed Edge Cover (MEC) to match clusters across a pair of samples. MEC allows a cluster to be matched with zero, one or more than one clusters in a paired sample. The cost of an MEC solution is equal to the summation of dissimilarities of the matched clusters and penalty for the unmatched clusters. The MEC algorithm finds an optimal solution by minimizing the cost of MEC, which is then used as dissimilarity between a pair of samples.

Value

`dist.sample` returns a numeric value representing dissimilarity between a pair of samples. This value is equal to the summation of dissimilarities of the matched clusters and penalty for the unmatched clusters.

Author(s)

Ariful Azad

References

Azad, Ariful and Langguth, Johannes and Fang, Youhan and Qi, Alan and Pothen, Alex (2010), Identifying rare cell populations in comparative flow cytometry; Algorithms in Bioinformatics, Springer, 162-175.

See Also

`ClusteredSample`, `match.clusters`

Examples

```
## ------------------------------------------------
## load data and retrieve two samples
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample1 = exprs(hd.flowSet[[1]])
sample2 = exprs(hd.flowSet[[2]])

## ------------------------------------------------
## cluster sample using kmeans algorithm
## ------------------------------------------------

clust1 = kmeans(sample1, centers=4, nstart=20)
clust2 = kmeans(sample2, centers=4, nstart=20)
cluster.labels1 = clust1$cluster
cluster.labels2 = clust2$cluster

## ------------------------------------------------
## Create ClusteredSample object
## and compute dissimilarity between two clustered samples
## using the mixed edge cover algorithm
## ------------------------------------------------

clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
clustSample2 = ClusteredSample(labels=cluster.labels2, sample=sample2)
D = dist.sample(clustSample1, clustSample2, dist.type='Mahalanobis', unmatch.penalty=999999)
```

`dist.template` *Dissimilarity between a pair of FC templates*

Description

Compute the dissimilarity between a pair of FC templates by using Mixed Edge Cover (MEC) algorithm.

Usage

```
dist.template(template1, template2, dist.type='Mahalanobis', unmatch.penalty=999999)
```

Arguments

`template1` an object of class `Template` containing cell populations from template 1.

`template2` an object of class `Template` containing cell populations from template 2.

`dist.type` character, indicating the method with which the dissimilarity between a pair of meta-clusters is computed. Supported dissimilarity measures are: 'Mahalanobis', 'KL' and 'Euclidean'.

`unmatch.penalty` A numeric value denoting the penalty for leaving a meta-cluster unmatched. This parameter should be already known or be estimated empirically estimated from data (see the reference for a discussion). Default is set to a very high value so that no meta-cluster remains unmatched.

Details

We used a robust version of matching called Mixed Edge Cover (MEC) to match meta-clusters across a pair of templates. MEC allows a meta-cluster to be matched with zero, one or more than one meta-clusters in a paired template. The cost of an MEC solution is equal to the summation of dissimilarities of the matched meta-clusters and penalty for the unmatched meta-clusters. The MEC algorithm finds an optimal solution by minimizing the cost of MEC, which is then used as dissimilarity between a pair of templates.

Value

`dist.template` returns a numeric value representing dissimilarity between a pair of templates. This value is equal to the summation of dissimilarities of the matched meta-clusters and penalty for the unmatched meta-clusters.

Author(s)

Ariful Azad

References

Azad, Ariful and Langguth, Johannes and Fang, Youhan and Qi, Alan and Pothen, Alex (2010), Identifying rare cell populations in comparative flow cytometry; Algorithms in Bioinformatics, Springer, 162-175.

See Also

Template, match.clusters

Examples

```
## ------------------------------------------------
## load data and retrieve two templates
## ------------------------------------------------

library(healthyFlowData)
data(hd)

## ------------------------------------------------
## Retrieve each sample, clsuter it and store the
## clustered samples in a list
## ------------------------------------------------

cat('Clustering samples: ')
clustSamples = list()
for(i in 1:10) # read 10 samples and cluster them
{
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
}

## ------------------------------------------------
## Create two templates each from five samples
## ------------------------------------------------

template1 = create.template(clustSamples[1:5])
template2 = create.template(clustSamples[6:10])

D = dist.template(template1, template2, dist.type='Mahalanobis', unmatch.penalty=999999)
```

---

mahalanobis.dist *Mahanalobis Distance*

Description

Compute the Mahalanobis distance between a pair of normally distributed clusters.

Usage

```
mahalanobis.dist(mean1, mean2, cov1, cov2, n1, n2)
```

Arguments

mean1	mean vector of length p for cluster 1, where p is the dimension of the clusters.
mean2	mean vector of length p for cluster 2.
cov1	pxp covariance matrix for cluster 1.
cov2	pxp covariance matrix for cluster 2.
n1	number of cells (points) in cluster 1.
n2	number of cells (points) in cluster 2.

Details

Consider two p-dimensional, normally distributed clusters with centers $\mu 1, \mu 2$ and covariance matrices $\Sigma 1, \Sigma 2$. Assume the size of the clusters are n1 and n2 respectively. We compute the Mahalanobis distance d12 between the clusters as follows:

$$\Sigma = ((n1 - 1) * \Sigma 1 + (n2 - 1) * \Sigma 2)/(n1 + n2 - 2)$$

$$d12 = sqrt(t(\mu 1 - \mu 2) * \Sigma^{(} - 1) * (\mu 1 - \mu 2))$$

The dimension of the clusters must be same.

Value

mahalanobis.dist returns a numeric value measuring the Mahalanobis distance between a pair of normally distributed clusters.

Author(s)

Ariful Azad

References

McLachlan, GJ (1999) Mahalanobis distance; Journal of Resonance 4(6), 20–26.

See Also

symmetric.KL, dist.cluster

Examples

```
## ------------------------------------------------
## load data and retrieve a sample
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample = exprs(hd.flowSet[[1]])


## ------------------------------------------------
## cluster sample using kmeans algorithm
## ------------------------------------------------

km = kmeans(sample, centers=4, nstart=20)
cluster.labels = km$cluster

## ------------------------------------------------
## Create ClusteredSample object
## and compute mahalanobis distance between two clsuters
## ------------------------------------------------

clustSample = ClusteredSample(labels=cluster.labels, sample=sample)
mean1 = get.center(get.clusters(clustSample)[[1]])
mean2 = get.center(get.clusters(clustSample)[[2]])
cov1 = get.cov(get.clusters(clustSample)[[1]])
cov2 = get.cov(get.clusters(clustSample)[[2]])
n1 = get.size(get.clusters(clustSample)[[1]])
n2 = get.size(get.clusters(clustSample)[[2]])
mahalanobis.dist(mean1, mean2, cov1, cov2, n1, n2)
```

`match.clusters` *Matching of clusters/meta-clusters across FC samples/templates*

Description

This function computes a matching of cluster/meta-clusters across a pair of FC samples/templates. A cluster (meta-cluster) from a sample (template) can match to zero, one or more than one clusters (meta-clusters) in another sample (template).

Usage

```
match.clusters(object1, object2, dist.type='Mahalanobis', unmatch.penalty=999999)

match.clusters.dist(d.matrix,unmatch.penalty=999999)
```

Arguments

- `object1` — an object of class `ClusteredSample` or `Template`.
- `object2` — an object of class `ClusteredSample` or `Template`.
- `dist.type` — character, indicating the method with which the dissimilarity between a pair of clusters (meta-clusters) is computed. Supported dissimilarity measures are: 'Mahalanobis', 'KL' and 'Euclidean', with the default is set to 'Mahalanobis' distance.
- `d.matrix` — a matrix used only in the second defination (`match.clusters.dist`) of the function. `d.matrix` stores the dissimilarities between each pair of clusters (meta-clusters) across a pair of samples (templates) `S1` and `S2`. `(i,j)` entry of the matrix stores dissimilarity between `i-th` cluster (meta-cluster) from `S1` and the `j-th` cluster (meta-cluster) from `S2`. `d.matrix` can be computed using function `dist.matrix`
- `unmatch.penalty` — numeric value denoting the penalty for leaving a cluster (meta-cluster) unmatched. This parameter should be already known or be estimated empirically estimated from data (see the reference for a discussion). Default is set to a very high value so that no cluster (meta-cluster) remains unmatched.

Details

We used a robust version of matching called Mixed Edge Cover (MEC) to match clusters (meta-clusters) across a pair of samples (templates). MEC allows a cluster (meta-cluster) to be matched with zero, one or more than one clusters (meta-clusters) across a pair of samples (template). The cost of an MEC solution is equal to the summation of dissimilarities of the matched clusters (meta-clusters) and penalty for the unmatched clusters (meta-clusters). The MEC algorithm finds an optimal solution by minimizing the cost of MEC.

Value

`match.clusters` returns an object of class `ClusterMatch` representing matching of clusters (meta-clusters) across a pair of FC samples (templates). A cluster (meta-cluster) from a sample (template) can match to zero, one or more than one cluster (meta-clusters) in another sample (template).

Author(s)

Ariful Azad

References

Azad, Ariful and Langguth, Johannes and Fang, Youhan and Qi, Alan and Pothen, Alex (2010), Identifying rare cell populations in comparative flow cytometry; Algorithms in Bioinformatics, Springer, 162-175.

See Also

`dist.matrix`, `ClusteredSample`, `Template`

Examples

```
## ------------------------------------------------
## load data and retrieve two samples
## ------------------------------------------------

library(healthyFlowData)
data(hd)

## ***********************************************************************
## ************** first matching clusters across samples *****************
## ***********************************************************************

## ------------------------------------------------
## retrieve and cluster two samples using kmeans algorithm
## ------------------------------------------------
sample1 = exprs(hd.flowSet[[1]])
sample2 = exprs(hd.flowSet[[2]])

clust1 = kmeans(sample1, centers=4, nstart=20)
clust2 = kmeans(sample2, centers=4, nstart=20)
cluster.labels1 = clust1$cluster
cluster.labels2 = clust2$cluster

## ------------------------------------------------
## Create ClusteredSample object
## and compute mahalanobis distance between two clsuters
## ------------------------------------------------

clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
clustSample2 = ClusteredSample(labels=cluster.labels2, sample=sample2)
## compute the dissimilarity matrix
DM = dist.matrix(clustSample1, clustSample2, dist.type='Mahalanobis')

## ------------------------------------------------
## Computing matching of clusteres
## An object of class "ClusterMatch" is returned
## ------------------------------------------------

## directly from the ClusteredSample objects: approach 1
mec = match.clusters(clustSample1, clustSample2, dist.type="Mahalanobis", unmatch.penalty=99999)
## from the dissimilarity matrix: approach 2
mec = match.clusters.dist(DM, unmatch.penalty=99999)
## show the matching
summary(mec)


## ***********************************************************************
## ************** Now matching meta-clusters across templates ************
## ***********************************************************************

## ------------------------------------------------
## Retrieve each sample, clsuter it and store the
```

```
## clustered samples in a list
## ------------------------------------------------

cat('Clustering samples: ')
clustSamples = list()
for(i in 1:10) # read 10 samples and cluster them
{
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
}

## ------------------------------------------------
## Create two templates each from five samples
## ------------------------------------------------

template1 = create.template(clustSamples[1:5])
template2 = create.template(clustSamples[6:10])

## ------------------------------------------------
## Match meta-clusters across templates
## ------------------------------------------------

mec = match.clusters(template1, template2, dist.type="Mahalanobis", unmatch.penalty=99999)
summary(mec)

## ------------------------------------------------
## Another example of matching meta-clusters & clusters
## across a template and a sample
## ------------------------------------------------

mec = match.clusters(template1, clustSample1, dist.type="Mahalanobis", unmatch.penalty=99999)
summary(mec)
```

`MetaCluster-class` *MetaCluster: An S4 class representing a meta-cluster (collection of biologically similar clusters).*

Description

An object of class "`MetaCluster`" represents a collection of biologically similar clusters from a set of FC samples. A meta-cluster is formed by matching clusters across samples and merging the matched clusters. An object of class "`ClusteredSample`" stores the estimated parameter of the whole meta-cluster as well as a list of clusters participating in the meta-cluster.

Creating Object

An object of class `MetaCluster` is usually created when constructing an object of class `Template`. Unless you know exactly what you are doing, creating an object of class "`MetaCluster`" using `new` or using the constructor is discouraged.

An object of class "`MetaCluster`" can be created using the following constructor

`MetaCluster(clusters)` where the argument "clusters" is a list of object of class `Cluster` from which the meta-cluster is created.

Slots

An object of class "`MetaCluster`" contains the following slots:

The number of clusters in the meta-cluster.

`num.clusters` `clusters` A list of length `num.clusters` storing the clusters (cell populations) participating in this meta-cluster. Each cluster is stored as an object of class `Cluster`.

`size` Number of cells in the meta-cluster (summation of all cluster sizes).

`center` A numeric vector denoting the center of the meta-cluster.

`cov` A matrix denoting the covariances of the underlying normal distribution of the meta-cluster.

Accessors

All the slot accessor functions take an object of class `MetaCluster`. I show usage of the first accessor function. Other functions can be called similarly.

The number of cells in the meta-cluster(summation of all cluster sizes).

Usage: `get.size(object)`

here `object` is a `MetaCluster` object.

`get.size` `get.num.clusters` Returns the number of clusters in the meta-cluster.

`get.clusters` Returns the list of clusters (cell populations) participating in this meta-cluster. Each cluster is stored as an object of class `Cluster`.

`get.size` Returns the number of cells in the meta-cluster (summation of all cluster sizes).

`get.center` Returns the center of the meta-cluster.

`get.cov` Returns the covariances matrix of the meta-cluster.

Methods

show Display details about the `Metacluster` object.

summary Return descriptive summary for the `MetaCluster` object.

Usage: `summary(MetaCluster)`

plot We plot a meta-cluster as a contour plot of the distribution of the underlying clusters or the combined meta-cluster. We consider cells in clusters or in the meta-cluster are normally distributed and represent the distribution with ellipsoid. The axes of an ellipsoid is estimated from the eigen values and eigen vectors of the covariance matrix ("Applied Multivariate Statistical Analysis" by R. Johnson and D. Wichern, 5th edition, Prentice hall). We then plot the

bi-variate projection of the ellipsoid as 2-D ellipses.

Usage:

`plot(mc, alpha=.05, plot.mc=FALSE, ...)`

the arguments of the plot function are:

- `mc` An object of class `MetaCluster` for which the plot function is invoked.
- `alpha` (1-alpha)*100% quantile of the distribution of the clusters or meta-cluster is plotted.
- `plot.mc` TRUE/FALSE, when TRUE the functions draws contour of the combined meta-cluster and when FALSE the function draws the contours of the individual clusters.
- `...` Other usual plotting related parameters.

Author(s)

Ariful Azad

References

Azad, Ariful and Pyne, Saumyadipta and Pothen, Alex (2012), Matching phosphorylation response patterns of antigen-receptor-stimulated T cells via flow cytometry; BMC Bioinformatics, 13 (Suppl 2), S10.

See Also

`Cluster`, `Template`

Examples

```
## ------------------------------------------------
## load data
## ------------------------------------------------

library(healthyFlowData)
data(hd)

## ------------------------------------------------
## Retrieve each sample, cluster it and store the
## clustered samples in a list
## ------------------------------------------------

cat('Clustering samples: ')
clustSamples = list()
for(i in 1:length(hd.flowSet))
{
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
```

```
}

## ------------------------------------------------
## Create a template from the list of clustered samples and retrieve the meta-clusters
## ------------------------------------------------

template = create.template(clustSamples)
#retrieve meta-clusters from template
mc = get.metaClusters(template)[[1]]
summary(mc)
# plot all participating cluster in this meta-cluster
plot(mc)
# plot the outline of the combined  meta-cluster
plot(mc, plot.mc=TRUE)
```

---

symmetric.KL *Symmetrized Kullback-Leibler divergence*

Description

Compute the Symmetrized Kullback-Leibler divergence between a pair of normally distributed clusters.

Usage

```
symmetric.KL(mean1, mean2, cov1, cov2)
```

Arguments

mean1	mean vector of length p for cluster 1, where p is the dimension of the clusters.
mean2	mean vector of length p for cluster 2.
cov1	pxp covariance matrix for cluster 1.
cov2	pxp covariance matrix for cluster 2.

Details

Consider two p-dimensional, normally distributed clusters with centers $\mu 1, \mu 2$ and covariance matrices $\Sigma 1, \Sigma 2$. We compute the KL divergence d12 between the clusters as follows:

$$d12 = 1/4 * (t(\mu 2 - \mu 1) * (\Sigma 1^{(} - 1) + \Sigma 2^{(} - 1)) * (\mu 2 - \mu 1) + trace(\Sigma 1/\Sigma 2 + \Sigma 2/\Sigma 1) + 2p)$$

The dimension of the clusters must be same.

Note that KL-divergence is not symmetric in its original form. We converted it symmetric by averaging both way KL divergence. The symmetrized KL-divergence is not a metric because it does not satisfy triangle inequality.

Value

`symmetric.KL` returns a numeric value measuring the Symmetrized Kullback-Leibler divergence between a pair of normally distributed clusters.

Author(s)

Ariful Azad

References

Abou–Moustafa, Karim T and De La Torre, Fernando and Ferrie, Frank P (2010) Designing a Metric for the Difference between Gaussian Densities; Brain, Body and Machine, 57–70.

See Also

`mahalanobis.dist`, `dist.cluster`

Examples

```
## ------------------------------------------------
## load data and retrieve a sample
## ------------------------------------------------

library(healthyFlowData)
data(hd)
sample = exprs(hd.flowSet[[1]])


## ------------------------------------------------
## cluster sample using kmeans algorithm
## ------------------------------------------------

km = kmeans(sample, centers=4, nstart=20)
cluster.labels = km$cluster

## ------------------------------------------------
## Create ClusteredSample object
## and compute mahalanobis distance between two clsuters
## ------------------------------------------------

clustSample = ClusteredSample(labels=cluster.labels, sample=sample)
mean1 = get.center(get.clusters(clustSample)[[1]])
mean2 = get.center(get.clusters(clustSample)[[2]])
cov1 = get.cov(get.clusters(clustSample)[[1]])
cov2 = get.cov(get.clusters(clustSample)[[2]])
n1 = get.size(get.clusters(clustSample)[[1]])
n2 = get.size(get.clusters(clustSample)[[2]])
symmetric.KL(mean1, mean2, cov1, cov2)
```

Template-class *Template: An S4 class representing a template of a group of FC Samples.*

Description

An object of class "Template" summarizes a group of samples belonging to same biological-class with a class-template. A template is represented by a collection of meta-clusters (MetaCluster) created from samples of same class. An object of class "Template" therefore stores a list of MetaCluster objects and other necessary parameters.

Creating Object

An object of class "Template" can be created using the function create.template :

create.template(clustSamples, dist.type = "Mahalanobis", unmatch.penalty=999999, template.id = NA_integer_).

The arguments to the create.template function is described below:

- clustSamples: A list of ClusteredSample objects from which the template is created. The working examples describe how this objects are created by clustering FC samples.
- dist.type: character, indicating the method with which the dissimilarity between a pair of clusters is computed. Supported dissimilarity measures are: 'Mahalanobis', 'KL' and 'Euclidean'. If this argument is not passed then 'Mahalanobis' distance is used by default.
- unmatch.penalty: A numeric value denoting the penalty for leaving a cluster unmatched. This parameter should be already known or be estimated empirically estimated from data (see the reference for a discussion). Default is set to a very high value so that no cluster remains unmatched.
- template.id: integer, denoting the index of the template (relative to other template). Default is NA_integer_

Slots

num.metaclusters: The number of meta-clusters in the template.

metaClusters: A list of length num.metaclusters storing the meta-clusters. Each meta-cluster is stored as an object of class MetaCluster.

dimension: Dimensionality of the samples from which the template is created.

size: Number of cells in the template (summation of all meta-cluster sizes).

tree: A list (similar to an hclust object) storing the hierarchy of the samples in a template.

template.id: integer, denoting the index of the template (relative to other templates). Default is NA_integer_

Accessors

All the slot accessor functions take an object of class `Template`. I show usage of the first accessor function. Other functions can be called similarly.

Number of cells in the template (summation of all meta-cluster sizes).

Usage: `get.size(object)`

here `object` is a `Template` object.

`get.size`: `get.num.metaclusters`: Returns the number of meta-clusters in the template.

`get.metaClusters`: Returns a list of length `num.metaclusters` storing the meta-clusters. Each meta-cluster is stored as an object of class `MetaCluster`.

`get.dimension`: Returns the dimensionality of the samples from which the template is created.

`get.tree`: Returns a `hclust` object storing the hierarchy of the samples in a template.

`get.template.id`: Returns the index of the template (relative to other templates).

Methods

show Display details about the `Template` object.

summary Return descriptive summary for each `MetaCluster` of a Template.

Usage: `summary(Template)`

plot We plot a template as a collection of bivariate contour plots of its meta-clusters. To plot each meta-cluster we consider the clusters within the meta-cluster normally distributed and represent each cluster with an ellipsoid. The axes of an ellipsoid is estimated from the eigen values and eigen vectors of the covariance matrix of a cluster ("Applied Multivariate Statistical Analysis" by R. Johnson and D. Wichern, 5th edition, Prentice hall). We then plot the bivariate projection of the ellipsoid as 2-D ellipses.

Usage:

```
plot(template, alpha=.05, plot.mc=FALSE, color.mc=NULL, colorbysample=FALSE, ...)
```

the arguments of the plot function are:

- `template`: An object of class `Template` for which the plot function is invoked.
- `alpha`: (1-alpha)*100% quantile of the distribution of the clusters or meta-cluster is plotted.
- `plot.mc`: TRUE/FALSE, when TRUE the functions draws contour of the combined meta-cluster and when FALSE the function draws the contours of the individual clusters.
- `color.mc`: A character vector of length `num.metaclusters` denoting the colors to be used to draw the contours. The ith color of this vector is used to draw the ellipses denoting clusters in the ith meta-cluster or the combined ith meta-cluster (depending on the argument `plot.mc`). By default an empty vector is passed and then an arbitrary color is used to draw each meta-cluster.
- `colorbysample`: TRUE/FALSE, when TRUE the functions draws clusters from same samples in a single color and when FALSE the function draws meta-clusters in a single color.
- `...` : Other usual plotting related parameters.

template.tree Plot the hierarchy of samples established while creating the template-tree. See template.tree

Author(s)

Ariful Azad

References

Azad, Ariful and Pyne, Saumyadipta and Pothen, Alex (2012), Matching phosphorylation response patterns of antigen-receptor-stimulated T cells via flow cytometry; BMC Bioinformatics, 13 (Suppl 2), S10.

See Also

MetaCluster, ClusteredSample, create.template, template.tree

Examples

```
## ------------------------------------------------
## load data
## ------------------------------------------------

library(healthyFlowData)
data(hd)

## ------------------------------------------------
## Retrieve each sample, clsuter it and store the
## clustered samples in a list
## ------------------------------------------------
set.seed(1234) # for reproducable clustering
cat('Clustering samples: ')
clustSamples = list()
for(i in 1:length(hd.flowSet))
{
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
}

## ------------------------------------------------
## Create a template from the list of clustered samples and plot functions
## ------------------------------------------------

template = create.template(clustSamples)
summary(template)

## plot the tree denoting the hierarchy of the samples in a template
tree = template.tree(template)
```

```
## plot the template in terms of the meta-clusters
## option-1 (default): plot contours of each cluster of the meta-clusters
plot(template)

## option-2: plot contours of each cluster of the meta-clusters with defined color
plot(template, color.mc=c('blue','black','green3','red'))

## option-3: plot contours of the meta-clusters with defined color
plot(template, plot.mc=TRUE, color.mc=c('blue','black','green3','red'))


## option-4: plot contours of each cluster of the meta-clusters with different colors for different samples
plot(template, colorbysample=TRUE)
```

`template.tree` *Plot the hierarchy of samples established while creating a template-tree*

Description

All samples within a template are organized as binary tree. This function plots the hierarchy of samples established while creating a template-tree.

Usage

```
template.tree(object, ...)
```

Arguments

`object`	An object of class `Template`. The working examples describe how a template is created from a collection of FC samples.
`...`	Other usual plotting related parameters.

Value

Returns a tree object of class `hclust` storing the hierarchy of the samples in the template.

Author(s)

Ariful Azad

References

Azad, Ariful and Pyne, Saumyadipta and Pothen, Alex (2012), Matching phosphorylation response patterns of antigen-receptor-stimulated T cells via flow cytometry; BMC Bioinformatics, 13 (Suppl 2), S10.

See Also

Template, create.template

Examples

```
 ## ------------------------------------------------
## load data
## ------------------------------------------------

library(healthyFlowData)
data(hd)

## ------------------------------------------------
## Retrieve each sample, clsuter it and store the
## clustered samples in a list
## ------------------------------------------------
set.seed(1234) # for reproducable clustering
cat('Clustering samples: ')
clustSamples = list()
for(i in 1:length(hd.flowSet))
{
  cat(i, ' ')
  sample1 = exprs(hd.flowSet[[i]])
  clust1 = kmeans(sample1, centers=4, nstart=20)
  cluster.labels1 = clust1$cluster
  clustSample1 = ClusteredSample(labels=cluster.labels1, sample=sample1)
  clustSamples = c(clustSamples, clustSample1)
}

## ------------------------------------------------
## Create a template from the list of clustered samples and plot functions
## ------------------------------------------------

template = create.template(clustSamples)
summary(template)

## plot the tree denoting the hierarchy of the samples in a template
tree = template.tree(template)
```

Index