Package ‘optimalFlow’

September 18, 2019

Type Package

Title What the Package Does (Title Case)

Version 0.1.0

Author Hristo Inouzhe

Maintainer Hristo Inouzhe <`hristo.inouzhe@gmail.com`>

Description More about what it does (maybe more than one line)
Use four spaces when indenting paragraphs within the Description.

License Artistic-2.0

Encoding UTF-8

LazyData true

Depends dplyr, optimalFlowData, rlang (>= 0.4.0)

Imports transport, parallel, Rfast, robustbase, dbscan, randomForest, foreach, graphics, doParallel, stats

Suggests knitr, BiocStyle, ellipse, rmarkdown

VignetteBuilder knitr

biocViews Software, FlowCytometry, Technology

NeedsCompilation no

R topics documented:

`calcobj` *calcobj*

Description

Calculates tclust's objective function value.

Usage

```
calcobj(X, iter, pa)
```

Arguments

`X`	Points in a multidimiensional space.
`iter`	Current solution obtained by tclust_.
`pa`	Parameters for using in tclust_.

Value

An object used internally in tclust_ with the the value for tclust's objective function.

References

Fritz, H., Garcia-Escudero, L. A., & Mayo-Iscar, A. (2012). tclust: An r package for a trimming approach to cluster analysis. Journal of Statistical Software, 47(12), 1-26.

Examples

```
x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),
matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 ,Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )

iter = output3$iter
pa = output3$pa

## calcobj  obtains the objective function value for data,
## an input parameters and a solution, including assigment and parameters

iter_ = calcobj (X=x, iter=iter, pa=pa)
iter_$obj
```

---

costWasserMatchingEllipse
costWasserMatchingEllipse

Description

Calculates similarity distance based on 2-Wassertein distance between mixtures of multivariate normal distributions.

Usage

```
costWasserMatchingEllipse(test.cytometry, training.cytometries, equal.weights = FALSE)
```

Arguments

`test.cytometry`	A clusetering represented as a list of clusters. Each cluster is a list with elements mean, cov, weight and type.
`training.cytometries`	A list of clusterings with the same format as test.cytometry.
`equal.weights`	If True, weights assigned to every cluster in a partion are uniform (1/number of clusters) when calculating the similarity distance. If False, weights assigned to clusters are the proportions of points in every cluster compared to the total amount of points in the partition.

Value

Returns a vector representing the similarity distance between test.cytometry and the elements in training.cytometries.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
partition1 = list(list(mean = c(1,1), cov = diag(1,2), weight = 0.5, type = "1"),
list(mean = c(-1,-1), cov = diag(1,2), weight = 0.5, type = "2"))
partition2 = list(list(list(mean = c(1,-1), cov = diag(1,2),
weight = 0.5, type = "1"), list(mean = c(-1,1), cov = diag(1,2), weight = 0.5, type = "2")))
costWasserMatchingEllipse(partition1, partition2)
```

`distGaussian` *distGaussian*

Description

Computes the squared Wasserstein distance between two normal multivariate distributions.

Usage

```
distGaussian(N1,N2)
```

Arguments

`N1`	A multivariate normal distribution as list with elements mean and cov.
`N2`	A multivariate normal distribution as list with elements mean and cov.

Value

Wasserstein squared distance between normals.

Examples

```
distGaussian(list(mean = c(-1,-1), cov = diag(2,2)), list(mean = c(1,1), cov = diag(1,2)))
```

`distGaussianCov` *distGaussianCov*

Description

Computes the componente relative to the covariances of the wasserstein distance.

Usage

```
distGaussianCov(N1,N2)
```

Arguments

`N1`	A multivariate normal distribution as list with elements mean and cov.
`N2`	A multivariate normal distribution as list with elements mean and cov.

Value

Wasserstein squared distance between normals with the same mean.

Examples

```
distGaussianCov(list(mean = c(1,1), cov = diag(2,2)), list(mean = c(1,1), cov = diag(1,2)))
```

`distGaussianMean` *distGaussianMean*

Description

Computes the componente relative to the means of the wasserstein distance.

Usage

```
distGaussianMean(N1,N2)
```

Arguments

`N1`	A multivariate normal distribution as list with elements mean and cov.
`N2`	A multivariate normal distribution as list with elements mean and cov.

Value

A value equivalent to the squared euclidean distance between the means.

Examples

```
distGaussianMean(list(mean = c(-1,-1), cov = diag(1,2)), list(mean = c(1,1), cov = diag(1,2)))
```

dmnorm *dmnorm*

Description

The multivariate normal density.

Usage

```
dmnorm(X, mu, sigma)
```

Arguments

X	Multivariate points.
mu	Mean.
sigma	COvariance matrix.

Value

The value of the multivariate normal with mean mu and covariance sigma at the points in X.

Examples

```
## Multivariate normal density
## Gives Multivariate normal density values for given  mu  and sigma
x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
dmnorm (X=x,mu=c(0,0),sigma=diag(2))
```

estimationCellBarycenter *estimationCellBarycenter*

Description

Estimates a Wasserstein barycenter for a cluster type using a collection of partitions.

Usage

```
estimationCellBarycenter(cell, cytometries)
```

Arguments

cell	Name of the cluster of interes
cytometries	List of clusterings

Value

`mean`	Mean of the barycenter.
`cov`	Covariance of the barycenter.
`weight`	Weight associated to the barycenter.
`type`	Type of the cluster.

Examples

```
partition1 = list(list(mean = c(1,1), cov = diag(1,2), weight = 0.5, type = "1"),
list(mean = c(-1,-1), cov = diag(1,2), weight = 0.5, type = "2"))
partition2 = list(list(mean = c(1,-1), cov = diag(1,2), weight = 0.5, type = "1"),
list(mean = c(-1,1), cov = diag(1,2), weight = 0.5, type = "2"))
cytometries = list(partition1, partition2)
estimationCellBarycenter("1",cytometries)
```

`estimClustPar` *estimClustPar*

Description

Obtain the best values for the model parameters, given data, input parameters and an assigment.

Usage

```
estimClustPar(X, iter, pa)
```

Arguments

`X`	Points in a multidimiensional space.
`iter`	Current solution obtained by tclust_.
`pa`	Parameters for using in tclust_.

Value

An object used internally in tclust_ with the best model parameters.

References

Fritz, H., Garcia-Escudero, L. A., & Mayo-Iscar, A. (2012). tclust: An r package for a trimming approach to cluster analysis. Journal of Statistical Software, 47(12), 1-26.

Examples

```
## tclust_ is the function which obtain the clusters for tclust_H function

x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 , Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )

##restr.diffax <- function (iter, pa)
## Apply constraints to covariance matrices
iter=output3$iter
pa=output3$pa

## estimClustPar  obtains the best values for the parameters,
##given data, input parameters and an assigment.
output4=estimClustPar (X=x, iter, pa)
output4$center
output4$sigma
output4$cw
```

`estimCovCellGeneral` *estimCovCellGeneral*

Description

Estimation of mean and covariance for a label in a partition.

Usage

```
estimCovCellGeneral(cell, cytometry, labels, type = "standard", alpha = 0.85)
```

Arguments

`cell`	Labell of the clsuter of interest.
`cytometry`	Data of the partition, wthout labels.
`labels`	Labels of the partition.
`type`	How to estimate covariance matrices of a cluster. "standard" is for using cov(), while "robust" is for using robustbase::covMcd.
`alpha`	Only when type = "robust". Indicates the value of alpha in robustbase::covMcd.

Value

`mean`	Mean of the cluster.
`cov`	Covariance of the cluster.
`weight`	Weight associated to the cluster.
`type`	Type of the cluster.

Examples

```
estimCovCellGeneral("Basophils", Cytometry1[,1:10], Cytometry1[,11])
```

f1Score *f1Score*

Description

Calculates the F1 score fore each group in a partition.

Usage

```
f1Score(clustering, cytometry, noise.cells)
```

Arguments

`clustering`	The labels of the new classification.
`cytometry`	Data of the clustering, where the last variable are the original labels.
`noise.cells`	An array of labels to be considered as noise.

Value

A matrix where the first row is the F1 score, the seconr row is the Precision and the third row is the Recall.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
f1Score(dplyr::pull(Cytometry3[c(sample(1:250,250),251:(dim(Cytometry3)[1])),],11),
Cytometry3, noise.types)
```

f1ScoreVoting *f1ScoreVoting*

Description

Calculates the F1 score fore each group in a partition, when provided with a fuzzy classification.

Usage

```
f1ScoreVoting(voting, clustering, cytometry, nivel_sup, noise.cells)
```

Arguments

`voting`	A list where each entry is a vote on the respective label
`clustering`	Labels of the partition
`cytometry`	Data of the clustering, where the last variable are the original labels.
`nivel_sup`	level of tolerance for assigning hard clustering. Should be greater or equal than 1. Class A is assigned if class A >nivel_sup*Class B.
`noise.cells`	An array of labels to be considered as noise.

Value

A matrix where the first row is the F1 score, the seconr row is the Precision and the third row is the Recall.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
database = list(as.data.frame(Cytometry2)[which(match(Cytometry2$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry3)[which(match(Cytometry3$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry4)[which(match(Cytometry4$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry5)[which(match(Cytometry5$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry7)[which(match(Cytometry7$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry8)[which(match(Cytometry8$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry9)[which(match(Cytometry9$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry12)[which(match(Cytometry12$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry13)[which(match(Cytometry13$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry14)[which(match(Cytometry14$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry15)[which(match(Cytometry15$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry16)[which(match(Cytometry16$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry17)[which(match(Cytometry17$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry19)[which(match(Cytometry19$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry21)[which(match(Cytometry21$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),])


templates.optimalFlow = optimalFlowTemplates(database = database, templates.number = 5,
cl.paral = 1)


classification.optimalFlow = optimalFlowClassification(as.data.frame(Cytometry1)[
which(match(Cytometry1$`Population ID (name)`,c("Monocytes", "CD4+CD8-",
"Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0), 1:10],database, templates.optimalFlow,
classif.method = "matching", cost.function = "ellipses", cl.paral = 1)


f1ScoreVoting(classification.optimalFlow$cluster.vote, classification.optimalFlow$cluster,
as.data.frame(Cytometry1)[which(match(Cytometry1$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),], 1.01, noise.types)
```

`findClustAssig` *findClustAssig*

Description

Function for obtaining the assigment and trimming (mixture models and hard assigment).

Usage

```
findClustAssig(X, iter, pa)
```

Arguments

`X`	Points in a multidimiensional space.
`iter`	Current solution obtained by tclust_.
`pa`	Parameters for using in tclust_.

Value

An object used internally in tclust_ with the assignment of the input points.

References

Fritz, H., Garcia-Escudero, L. A., & Mayo-Iscar, A. (2012). tclust: An r package for a trimming approach to cluster analysis. Journal of Statistical Software, 47(12), 1-26.

Examples

```
##### tclust_ is the function which obtain the clusters for tclust_H function

x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 , Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )

iter = output3$iter
pa = output3$pa

output5 = findClustAssig(X=x, iter, pa)
```

GaussianBarycenters *GaussianBarycenters*

Description

Finds the barricenter of a mixture of normals with the same mean.

Usage

```
GaussianBarycenters(matrices, weight)
```

Arguments

matrices	A list of covariance matrices.
weight	A vector of weights associated to the covariance matrices.

Value

Barycenter	Returns the barycenter, a covariance matrix.
Variation	The Wasserstein Variation.
Num.iter	THe number of iterations to achieve the stopping criteria.

Examples

```
GaussianBarycenters(list(diag(2,2),diag(1,2)), c(0.5,0.5))
```

getini *getini*

Description

Calculates the initial cluster sizes for initializing tclust_.

Usage

```
getini(K, no.trim)
```

Arguments

K	Number of groups.
no.trim	Indicates if trimming is allowed.

Value

A vector of length K indicating indices of the points to be considered for initialization.

Examples

```
## gives a random vector from a K dimensional multinomial(no.trim, pi.ini)
## with pi.ini a random vector build with random values from uniform (0,1)
v=getini  (K=3, no.trim=100)
v
```

InitClusters *InitClusters*

Description

Calculates the initial cluster assignment and initial values for the parameters

Usage

```
InitClusters(X, iter, pa)
```

Arguments

X	Points in a multidimiensional space.
iter	Current solution obtained by tclust_.
pa	Parameters for using in tclust_.

Value

An initial solution, based on a random subsample, in a form of an object used internally in tclust_.

Examples

```
######EXAMPLE tclust_
##### tclust_ is the function which obtain the clusters for tclust_H function

x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 , Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )

##restr.diffax <- function (iter, pa)
## Apply constraints to covariance matrices
iter=output3$iter
pa=output3$pa

#Gives an initial solution based on a random subsample
iter=InitClusters (X=x, iter=output3$iter, pa=output3$pa)
iter$cw
iter$center
iter$sigma
```

---

kcenter *kcenter*

Description

Calculates the k-barycenter of a list on multivaraite normals for a given the number of clusters and an assignation of each normal to the respective group.

Usage

```
kcenter(points, kk, center.asigned)
```

Arguments

`points`	List of multivariate normals, where each element is a list with values mean and cov.
`kk`	The number k of groups for the k-barycenter.
`center.asigned`	A vector indicating to which group each normal should be assigned.

Value

`kcenters`	A list of the elements of the k-barycenter, i.e., a list of k lists containing means and covariances.
`t.variation`	The trimmed wasserstein variation.

Examples

```
normals = list(list(mean = c(1,1), cov = diag(2,2)),
list(mean = c(1,1), cov = diag(1,2)), list(mean = c(3,3), cov = diag(1,2)))
kcenter(normals, 2, c(1,1,2))
```

---

labelTransfer *labelTransfer*

Description

Label transfer between a test partition and a training set of partitions.

Usage

```
labelTransfer(training.cytometry, test.cytometry, test.partition, equal.weights = FALSE)
```

Arguments

`training.cytometry`	List of partitions, where each partition is a dataframe wher the last column contains the labels of the partition.
`test.cytometry`	Test data, a dataframe without labels.
`test.partition`	Labels of a partition of the test data.
`equal.weights`	If True, weights assigned to every cluster in a partion are uniform (1/number of clusters) when calculating the similarity distance. If False, weights assigned to clusters are the proportions of points in every cluster compared to the total amount of points in the partition.

Value

A fuzzy relabeling consistent of a transportation plan.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
data.example = data.frame(v1 = c(rnorm(50,2,1), rnorm(50,-2,1)),
v2 = c(rnorm(50,2,1), rnorm(50,-2,1)), id = c(rep(0,50), rep(1,50)))
test.labels = c(rep("a",50), rep("b", 50))
labelTransfer(data.example, data.example[,1:2], test.labels)
```

`labelTransferEllipse` *labelTransferEllipse*

Description

Label transfer between a test partition and a training partitions viewed as a mixture of gaussians.

Usage

```
labelTransferEllipse(i, test.cytometry.ellipses,
training.cytometries.barycenter, equal.weights = FALSE)
```

Arguments

`i`	A dummy variable, should be any integral. Ment for use with lapply.
`test.cytometry.ellipses`	A test clustering viewed as a mixture of multivariate normal distributions.
`training.cytometries.barycenter`	A training partition viewed as a mixture of multivariate normal distributions.
`equal.weights`	If True, weights assigned to every cluster in a partion are uniform (1/number of clusters) when calculating the similarity distance. If False, weights assigned to clusters are the proportions of points in every cluster compared to the total amount of points in the partition.

Value

A fuzzy relabeling consistent of a transportation plan.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
partition1 = list(list(mean = c(1,1), cov = diag(1,2), weight = 0.5, type = "1"),
list(mean = c(-1,-1), cov = diag(1,2), weight = 0.5, type = "2"))
partition2 = list(list(mean = c(1,1), cov = diag(1,2), weight = 0.5, type = "a"),
list(mean = c(-1,-1), cov = diag(1,2), weight = 0.5, type = "b"))
labelTransferEllipse(1, partition2, partition1)
```

optimalFlowClassification *optimalFlowClassification*

Description

Performs a supervised classification of the input data where a database and a partition of the database are available

Usage

```
optimalFlowClassification(X, database, templates, consensus.method = "pooling",
cov.estimation = "standard", alpha.cov = 0.85,initial.method = "supervized",
alpha.tclust = 0, restr.factor.tclust = 1000,classif.method = "qda",
qda.bar = TRUE, cost.function = "points", cl.paral = 1,
equal.weights.voting = TRUE, equal.weights.template = TRUE)
```

Arguments

`X`	Datasample to be classified.
`database`	A list where each entry is a partition (clustering) represented as dataframe, of the same dimensions, where the last variable represents the labels of the partition.
`templates`	List of the consensus clusterings for every group in the partition of the database obtained by optimalFlowTemplates
`consensus.method`	The consensus.method value that was used in optimalFlowTemplates.
`cov.estimation`	How to estimate covariance matrices in each cluster of a partition. "standard" is for using cov(), while "robust" is for using robustbase::covMcd.
`alpha.cov`	Only when cov.estimation = "robust". Indicates the value of alpha in robustbase::covMcd.
`initial.method`	Indicates how to initialize tclust. Currently only supports "supervised".
`alpha.tclust`	Level of trimming allowed fo tclust.
`restr.factor.tclust`	Fixes the restr.fact parameter in tclust.
`classif.method`	Indicates what type of supervised learning we want to do. Takes values on c("matching", "qda", "random forest").
`qda.bar`	Only if classif.method = "qda". If True then the appropriate consensus clustering (template, prototype) is used for learning. If False, the closest partition in the appropriate group is used.
`cost.function`	Only if classif.method = "matching". Indicates the cost function, distance between clusters, to be used for label matching.
`cl.paral`	Number of cores to be used in parallel procedures.

`equal.weights.voting`
: only when classif.method = "qda" and qda.bar =F, or when classif.method = "random forest". Indicates the weights structure when looking for the most similar partition in a group.

`equal.weights.template`
: If True, weights assigned to every cluster in a partion are uniform (1/number of clusters). If False, weights assigned to clusters are the proportions of points in every cluster compared to the total amount of points in the partition.

Value

`cluster`	Labels assigned to the input data.
`clusterings`	The different partitions of the data obtained by tclust when initialized with the templates obtained by optimalFlowTemplates.
`assigned.template.index`	Label of the group for which the template is closer to the data.
`cluster.vote`	Only when classif.method = "matching". Vote on the type of every label in the partition of the data.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
database = list(as.data.frame(Cytometry2)[which(match(Cytometry2$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry3)[which(match(Cytometry3$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry4)[which(match(Cytometry4$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry5)[which(match(Cytometry5$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry7)[which(match(Cytometry7$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry8)[which(match(Cytometry8$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry9)[which(match(Cytometry9$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry12)[which(match(Cytometry12$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry13)[which(match(Cytometry13$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry14)[which(match(Cytometry14$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry15)[which(match(Cytometry15$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry16)[which(match(Cytometry16$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry17)[which(match(Cytometry17$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry19)[which(match(Cytometry19$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry21)[which(match(Cytometry21$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),])


templates.optimalFlow = optimalFlowTemplates(database = database, templates.number = 5,
cl.paral = 1)


classification.optimalFlow = optimalFlowClassification(as.data.frame(Cytometry1)[
which(match(Cytometry1$`Population ID (name)`,c("Monocytes", "CD4+CD8-", "Mature SIg Kappa",
"TCRgd-"), nomatch = 0)>0), 1:10], database, templates.optimalFlow, cl.paral = 1)


scoreF1.optimalFlow = optimalFlow::f1Score(classification.optimalFlow$cluster,
as.data.frame(Cytometry1)[which(match(Cytometry1$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),], noise.types)
```

`optimalFlowTemplates` *optimalFlowTemplates*

Description

Returns a partition of the input clusterings with a respective consensus cluster for every group in the partition.

Usage

```
optimalFlowTemplates(database, database.names = NULL, cov.estimation = "standard",
alpha.cov = 0.85,equal.weights.template = TRUE, hclust.method = "complete",
templates.number = NA,minPts = 2, eps = 1, consensus.method = "pooling",
barycenters.number = 37,bar.repetitions = 40, alpha.bar = 0.05,
bar.ini.method = "plus-plus", consensus.minPts = 3, cl.paral = 1)
```

Arguments

`database`	A list where each entry is a partition (clustering) represented as dataframe, of the same dimensions, where the last variable represents the labels of the partition.
`database.names`	Names of the elements in the database.
`cov.estimation`	How to estimate covariance matrices in each cluster of a partition. "standard" is for using cov(), while "robust" is for using robustbase::covMcd.
`alpha.cov`	Only when cov.estimation = "robust". Indicates the value of alpha in robustbase::covMcd.

`equal.weights.template`	If True, weights assigned to every cluster in a partion are uniform (1/number of clusters). If False, weights assigned to clusters are the proportions of points in every cluster compared to the total amount of points in the partition.
`hclust.method`	Indicates what kind of hierarchical clustering to do with the similarity distances matrix of the partitions. Takes values in c("complete", "single", "average", "hdbscan", "dbscan").
`templates.number`	Only if hclust.method in c("complete", "single", "average"). Indicates the number of clusters to use with cutree. If set to NA (default), plots the hierarchical tree and asks the user to introduce an appropriate number of clusters.
`minPts`	Only if hclust.method in c("hdbscan", "dbscan"). Indicates the value of argument minPts in dbscan::dbscan and dbscan::hdbscan.
`eps`	Only if hclust.method = "dbscan". Indicates the value of eps in dbscan::dbscan.
`consensus.method`	Sets the way of doing consensus clustering when clusters are viewed as Multivariate Distributions. Can take values in c("pooling", "k-barycenter", "hierarchical"). See details.
`barycenters.number`	Only if consensus.method = "k-barycenter". Sets the number, k, of barycenters when using k-barycenters.
`bar.repetitions`	Only if consensus.method = "k-barycenter". How many times to repeat the k-barycenters procedure. Equivalent to nstart in kmeans.
`alpha.bar`	Only if consensus.method = "k-barycenter". The level of trimming allowed during the k-barycenters procedure.
`bar.ini.method`	Only if consensus.method = "k-barycenter". Takes values in c("rnd", "plusplus"). See details.
`consensus.minPts`	Only if consensus.method = "hierarchical". The value of argument minPts for dbscan::hdbscan.
`cl.paral`	Number of cores to be used in parallel procedures.

Value

`templates`	A list of the consensus clusterings for every group in the partition of the database.
`clustering`	Clustering of the input partitions.
`database.elliptical`	Means, covariances and weights of the clusters in the input partitions.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
database = list(as.data.frame(Cytometry2)[which(match(Cytometry2$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry3)[which(match(Cytometry3$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry4)[which(match(Cytometry4$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry5)[which(match(Cytometry5$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry7)[which(match(Cytometry7$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry8)[which(match(Cytometry8$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry9)[which(match(Cytometry9$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry12)[which(match(Cytometry12$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry13)[which(match(Cytometry13$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry14)[which(match(Cytometry14$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry15)[which(match(Cytometry15$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry16)[which(match(Cytometry16$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry17)[which(match(Cytometry17$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry19)[which(match(Cytometry19$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry21)[which(match(Cytometry21$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),])


templates.optimalFlow = optimalFlowTemplates(database = database, templates.number = 5,
cl.paral = 1)
```

`qdaClassification` *qdaClassification*

Description

GIves quadratic discriminant scores to the poiints in data for a multivariate normal.

Usage

```
qdaClassification(normal, data)
```

Arguments

`normal`	A list with arguments mean, covaruance and weight.
`data`	Data frame or matrix on which to perform qda.

Value

A score for each point.

Examples

```
data.qda = cbind(rnorm(50), rnorm(50))
exp(qdaClassification(list(mean = c(0,0), cov = diag(1,2), weight = 1), data.qda))
```

`restr.diffax` *restr.diffax*

Description

Ment for applying restrictions to solutions obtained by tclust_.

Usage

```
restr.diffax(iter, pa)
```

Arguments

`iter`	Current solution obtained by tclust_.
`pa`	Parameters for using in tclust_.

Value

The neew restricted solution.

Examples

```
######EXAMPLE tclust_
##### tclust_ is the function which obtain the clusters for tclust_H function

x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 , Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )

##restr.diffax <- function (iter, pa)
## Apply constraints to covariance matrices
iter=output3$iter
pa=output3$pa
```

```
pa$maxfact_e=1.1
print(iter$sigma)
iter2=restr.diffax (iter, pa)
print(iter2$sigma)
```

---

restr2_eigenv *restr2_eigenv*

Description

Function for applying eigenvalue constraints. These are the typical constraints used in tclust.

Usage

```
restr2_eigenv(autovalues, ni.ini, factor_e, zero.tol)
```

Arguments

autovalues	Matrix containin eigenvalues.
ni.ini	Current sample size of the clusters.
factor_e	The level of the constraints.
zero.tol	Toletance level.

Value

The restricted eigenvalues

References

Fritz, H., Garcia-Escudero, L. A., & Mayo-Iscar, A. (2012). tclust: An r package for a trimming approach to cluster analysis. Journal of Statistical Software, 47(12), 1-26.

Examples

```
#restr2_eigenv <- function(autovalues, ni.ini, factor_e, zero.tol)
#gives optimal constrained eigenvalues
autovalues=matrix(c(2,3,4,1,2,3),nrow=2)
ni.ini=c(2,2,3)
factor_e=1.1
zero.tol=1e-9
autovalues_const= restr2_eigenv (autovalues, ni.ini, factor_e, zero.tol)
autovalues_const
```

ssclmat *ssclmat*

Description

Extract a matrix from the object containing covariance matrices. Ment for use inside tclust_ function.

Usage

```
ssclmat(x, k)
```

Arguments

x	An object used inside tclust_ containing covariance matrices.
k	An integer value giving a location in an array.

Value

A covariance matrix.

Examples

```
######EXAMPLE tclust_
##### tclust_ is the function which obtain the clusters for tclust_H function

x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 , Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )

##restr.diffax <- function (iter, pa)
## Apply constraints to covariance matrices
iter=output3$iter
pa=output3$pa
pa$maxfact_e=1.1
iter2=restr.diffax (iter, pa)

##EXAMPLE extract matrix from the object containing covariance matrices
##sclmat <- function (x, k) as.matrix (x[,,k])
ssclmat(iter2$sigma,k=1)
```

tclustWithInitialization *tclustWithInitialization*

Description

A wrapper for the function tclust_H.

Usage

```
tclustWithInitialization(initialization, cytometry, i.sol.type = "points",
trimming = 0.05, restr.fact = 1000)
```

Arguments

`initialization`	Initial solution for parameters provided by the user. Can be a matrix of data containing observations anc cluster assignations or can be a list spesifying a multivariate mixture of gaussians.
`cytometry`	A matrix or data.frame of dimension n x p, containing the observations (row-wise).
`i.sol.type`	Type of initial solutions in c("points", "barycenters"). "points" refers to a classified data matrix, while "barycenters" to a multivariate mixture.
`trimming`	The proportion of observations to be trimmed.
`restr.fact`	The constant restr.fact >= 1 constrains the allowed differences among group scatters. Larger values imply larger differences of group scatters, a value of 1 specifies the strongest restriction.

Value

`cluster`	A numerical vector of size n containing the cluster assignment for each observation. Cluster names are integer numbers from 1 to k, 0 indicates trimmed observations.
`n_clus`	Number of clusters actually found.
`obj`	he value of the objective function of the best (returned) solution.

References

Fritz, H., Garcia-Escudero, L. A., & Mayo-Iscar, A. (2012). tclust: An r package for a trimming approach to cluster analysis. Journal of Statistical Software, 47(12), 1-26. E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006.

Examples

```
x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),
matrix(rnorm(100)+4,ncol=2))

## robust cluster obtention from a sample x asking for 3 clusters,
## trimming level 0.05 and constrain level 12

k=3;alpha = 0.05;restr.fact = 12
output=tclust_H (x=x, k = k, alpha = alpha, nstart = 50, iter.max = 20,
restr = "eigen", restr.fact = restr.fact, sol_ini_p = FALSE, sol_ini = NA,
equal.weights = FALSE, center = center, scale = scale, store.x = TRUE,
drop.empty.clust = TRUE, trace = 0, warnings = 3, zero.tol = 1e-16)

## cluster assigment

output2 = tclustWithInitialization(data.frame(x, output$cluster), x, "points", 0.05, 10)
```

`tclust_` *tclust_*

Description

Function that performs robust non spherical clustering, tclust, where initial solutions are allowed.

Usage

```
tclust_(X, K, alpha = 0.05, niter = 20, Ksteps = 10, equal.weights = FALSE,
restr.cov.value = "eigen", maxfact_e = 5, zero.tol = 1e-16, trace = 0,
sol_ini_p = FALSE, sol_ini = NA)
```

Arguments

`X`	Points in a multidimiensional space.
`K`	Number of clusters.
`alpha`	Level of trimming.
`niter`	Maximum number of iterations.
`Ksteps`	Maximum number of K-steps.
`equal.weights`	If all clusters should have equal weight in the objective function.
`restr.cov.value`	Type of restriction on covariance matrices. Only "eigen" will be allowed.
`maxfact_e`	Level determinant constraints.
`zero.tol`	The zero tolerance used. By default set to 1e-16..
`trace`	Defines the tracing level, which is set to 0 by default. Tracing level 2 gives additional information on the iteratively decreasing objective function's value.
`sol_ini_p`	Initial solution for parameters provided by the user TRUE/FALSE, if TRUE is stored in sol_ini.
`sol_ini`	Initial solution for parameters provided by the user.

Value

iter	Contains the solutions of the clustering provided by tclust_.
pa	Parametaers used in tslucs_.
X	The input variables.

References

Fritz, H., Garcia-Escudero, L. A., & Mayo-Iscar, A. (2012). tclust: An r package for a trimming approach to cluster analysis. Journal of Statistical Software, 47(12), 1-26.

Examples

```
x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 , Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )
## cluster assigment
output3$iter$assig
plot(x,col=output3$iter$assig+1)
```

---

tclust_H *tclust_H*

Description

A wrapper for the fucntion tclust_. Performs robust non spherical clustering, tclust, where initial solutions are allowed.

Usage

```
tclust_H(x, k = 3, alpha = 0.05, nstart = 50, iter.max = 20, restr = "eigen",
restr.fact = 12, sol_ini_p = FALSE, sol_ini = NA, equal.weights = FALSE,
center = center, scale = scale, store.x = TRUE, drop.empty.clust = TRUE,
trace = 0, warnings = 3, zero.tol = 1e-16)
```

Arguments

x	A matrix or data.frame of dimension n x p, containing the observations (row-wise).
k	The number of clusters initially searched for.
alpha	The proportion of observations to be trimmed.
nstart	The number of random initializations to be performed. Only when sol_ini_p = FALSE.
iter.max	The maximum number of concentration steps to be performed. The concentration steps are stopped, whenever two consecutive steps lead to the same data partition.

`restr`	The type of restriction to be applied on the cluster scatter matrices. Valid values are "eigen" (default).
`restr.fact`	The constant restr.fact >= 1 constrains the allowed differences among group scatters. Larger values imply larger differences of group scatters, a value of 1 specifies the strongest restriction.
`sol_ini_p`	Initial solution for parameters provided by the user TRUE/FALSE, if TRUE is stored in sol_ini.
`sol_ini`	Initial solution for parameters provided by the user.
`equal.weights`	A logical value, specifying whether equal cluster weights (TRUE) or not (FALSE) shall be considered in the concentration and assignment steps.
`center`	A center vector of length p which can optionally be specified for centering x before calculation.
`scale`	A scale vector of length p which can optionally be specified for scaling x before calculation.
`store.x`	A logical value, specifying whether the data matrix x shall be included in the result structure. By default this value is set to TRUE.
`drop.empty.clust`	Logical value specifying, whether empty clusters shall be omitted in the resulting object. (The result structure does not contain center and covariance estimates of empty clusters anymore. Cluster names are reassigned such that the first l clusters (l <= k) always have at least one observation.
`trace`	Defines the tracing level, which is set to 0 by default. Tracing level 2 gives additional information on the iteratively decreasing objective function's value.
`warnings`	The warning level (0: no warnings; 1: warnings on unexpected behavior; 2: warnings if restr.fact causes artificially restricted results).
`zero.tol`	The zero tolerance used. By default set to 1e-16.

Details

This iterative algorithm initializes k clusters randomly and performs "concentration steps" in order to improve the current cluster assignment. The number of maximum concentration steps to be performed is given by iter.max. For approximately obtaining the global optimum, the system is initialized nstart times and concentration steps are performed until convergence or iter.max is reached. When processing more complex data sets higher values of nstart and iter.max have to be specified (obviously implying extra computation time). However, if more then half of the iterations would not converge, a warning message is issued, indicating that nstart has to be increased.

The parameter restr defines the cluster's shape restrictions, which are applied on all clusters during each iteration. Options "eigen"/"deter" restrict the ratio between the maximum and minimum eigenvalue/determinant of all cluster's covariance structures to parameter restr.fact. Setting restr.fact to 1, yields the strongest restriction, forcing all eigenvalues/determinants to be equal and so the method looks for similarly scattered (respectively spherical) clusters. Option "sigma" is a simpler restriction, which averages the covariance structures during each iteration (weighted by cluster sizes) in order to get similar (equal) cluster scatters.

Value

`centers`	A matrix of size p x k containing the centers (column-wise) of each cluster.
`cov`	An array of size p x p x k containing the covariance matrices of each cluster.
`cluster`	A numerical vector of size n containing the cluster assignment for each observation. Cluster names are integer numbers from 1 to k, 0 indicates trimmed observations.
`par`	A list, containing the parameters the algorithm has been called with (x, if not suppressed by store.x = FALSE, k, alpha, restr.fact, nstart, KStep, and equal.weights).
`weights`	A numerical vector of length k, containing the weights of each cluster.
`obj`	he value of the objective function of the best (returned) solution.

References

Fritz, H., Garcia-Escudero, L. A., & Mayo-Iscar, A. (2012). tclust: An r package for a trimming approach to cluster analysis. Journal of Statistical Software, 47(12), 1-26.

Examples

```
## tclust_H if the function which gives clusters to the user.
## The main role of this function is to be an interface with
## the user using labels for the parameters similar to tclust
## function in tclust package

x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),
matrix(rnorm(100)+4,ncol=2))

## robust cluster obtention from a sample x asking for 3 clusters,
## trimming level 0.05 and constrain level 12

k=3;alpha = 0.05;restr.fact = 12
output=tclust_H (x=x, k = k, alpha = alpha, nstart = 50, iter.max = 20,
restr = "eigen", restr.fact = restr.fact, sol_ini_p = FALSE, sol_ini = NA,
equal.weights = FALSE, center = center, scale = scale, store.x = TRUE,
drop.empty.clust = TRUE, trace = 0, warnings = 3, zero.tol = 1e-16)

## cluster assigment

output$cluster
plot(x,col=output$cluster)
```

`TreatSingularity` *TreatSingularity*

Description

Shows a warning message when a singularity in the curent solution of tclust_ is found.

Usage

```
TreatSingularity(iter, pa)
```

Arguments

`iter`	Current solution obtained by tclust_
`pa`	Parameters for using in tclust_.

Value

A warning message.

Examples

```
######EXAMPLE tclust_
##### tclust_ is the function which obtain the clusters for tclust_H function

x=rbind(matrix(rnorm(100),ncol=2),matrix(rnorm(100)+2,ncol=2),matrix(rnorm(100)+4,ncol=2))
output3=tclust_( X=x , K=3 , alpha = 0.05 , niter = 20 , Ksteps=10 ,
equal.weights = FALSE, restr.cov.value = "eigen" ,
maxfact_e = 5 , zero.tol = 1e-16 ,  trace = 0 ,
sol_ini_p = FALSE ,   sol_ini=NA )

##restr.diffax <- function (iter, pa)
## Apply constraints to covariance matrices
iter=output3$iter
pa=output3$pa

##### It shows a warning message
##### warning ("points in the data set are concentrated in k points after trimming ")
TreatSingularity (iter, pa)
```

`trimmedKBarycenter` *trimmedKBarycenter*

Description

Calculates K-barycenters of multivariate normal distributions with the 2-Wasserstein distance

Usage

```
trimmedKBarycenter(k, alpha0, type.ini = "rnd", reps.list)
```

Arguments

`k`	Number k of elements in the k-barycenter.
`alpha0`	Level of trimming.
`type.ini`	Type of initialization in c("rnd", "plus-plus"). "rnd" makes the common random initilaization while "plus-plus" initializes in a similar fashion to k-means++.
`reps.list`	List of multivariate normals for which the trimmed k-barycenter should be performed.

Value

`variacion_wasser`	Waserstein variation.
`baricentro`	A list of k elements, each of which is a member of the k-barycenter
`cluster`	The assignment of the original entries to each member of the k-barycenter.

Examples

```
normals = list(list(mean = c(1,1), cov = diag(2,2)), list(mean = c(1,1),
cov = diag(1,2)), list(mean = c(3,3), cov = diag(1,2)))
trimmedKBarycenter(2, 0, "rnd", normals)
```

`trimmedMinDist` *trimmedMinDist*

Description

For two lists of multivaraite normals, points and centers, returns the index of which element in centers is closest to each element in points when some trimming is allowed.

Usage

```
trimmedMinDist(points, centres, alpha = 0.1)
```

Arguments

`points`	List of multivariate normals, where each element is a list with values mean and cov.
`centres`	List of multivariate normals, where each element is a list with values mean and cov.
`alpha`	Level of triming

Value

A vector with the index of which element in centers is closest to each element in points.

Examples

```
normals = list(list(mean = c(1,1), cov = diag(2,2)),
list(mean = c(1,1), cov = diag(1,2)), list(mean = c(3,3), cov = diag(1,2)))
k_barycenter = kcenter(normals, 2, c(1,1,2))$kcenters
trimmedMinDist(normals,k_barycenter, 0)
```

`voteLabelTransfer` *voteLabelTransfer*

Description

A wrapper for doing either labelTransfer or labelTransferEllipse

Usage

```
voteLabelTransfer(type = "points", test.partition, test.cytometry,
test.partition.ellipse, training.cytometries,training.cytometries.barycenter,
test = 1, op.syst, cl.paral = 1, equal.weights = FALSE)
```

Arguments

`type`	"points" indicates use of labelTransfer; "ellipses" of labelTransferEllipse.
`test.partition`	Only when type = "points". Labels of a partition of the test data.
`test.cytometry`	Only when type = "points". Test data, a dataframe without labels.
`test.partition.ellipse`	Only when type = "ellipses". A test clustering viewed as a mixture of multivariate normal distributions.
`training.cytometries`	Only when type = "points". List of partitions, where each partition is a dataframe wher the last column contains the labels of the partition.
`training.cytometries.barycenter`	Only when type = "ellipses". A training partition viewed as a mixture of multivariate normal distributions.
`test`	Only when type = "ellipses". A dummy variable, should be any integral. Ment for use with lapply.
`op.syst`	Type of system, takes values in c("unix", "windows").
`cl.paral`	Number of cores to be used in parallel procedures.
`equal.weights`	If True, weights assigned to every cluster in a partion are uniform (1/number of clusters) when calculating the similarity distance. If False, weights assigned to clusters are the proportions of points in every cluster compared to the total amount of points in the partition.

Value

`final.vote`	A list for the votes on each cell.
`complete.vote`	A more complete list for the votes on each cell.

Examples

```
data.example = data.frame(v1 = c(rnorm(50,2,1), rnorm(50,-2,1)),
v2 = c(rnorm(50,2,1), rnorm(50,-2,1)), id = c(rep(0,50), rep(1,50)))
test.labels = c(rep("a",50), rep("b", 50))
voteLabelTransfer(test.partition = test.labels, test.cytometry = data.example[,1:2],
training.cytometries = list(data.example), op.syst = .Platform$OS.type)$final.vote[[1]]
```

`voteTransformation` *voteTransformation*

Description

Transforming votes obtained by using optimalFlowTemplates + OptimalFlowClassification with consenus.method in c("hierarchical", "k-barycenter") and classif.method = "matching" and cost.function = "ellipses" to an appropriate format for using f1ScoreVoting.

Usage

```
voteTransformation(vote.0, vote.1)
```

Arguments

`vote.0`	Values obtained by voteLabelTransfer
`vote.1`	Original proportions of the clusters after the template obtention.

Value

A list for the votes on each cell.

Examples

```
vote.0 = list("1" = data.frame(cell = c(1,2), "compound.proportion" = c(0.7,0.3),
"simple.proportion"= c(0.7,0.3)), "2" = data.frame(cell = c(1,2),
"compound.proportion" = c(0.3,0.7), "simple.proportion"= c(0.3,0.7)))
vote.1.1 = t(c(0.8,0.2))
names(vote.1.1) = c("A","B")
vote.1.2 = t(c(0.2,0.8))
names(vote.1.2) = c("A","B")
vote.1 = list(vote.1.1, vote.1.2)
voteTransformation(vote.0, vote.1)
```

w2dist *w2dist*

Description

The 2-Wasserstein distance between two multivariate normal distributions

Usage

```
w2dist(P,Q)
```

Arguments

`P`	A multivariate normal distribution given as a list with arguments mean and cov.
`Q`	A multivariate normal distribution given as a list with arguments mean and cov.

Value

The 2-Wasserstein distance between the two distributions.

Examples

```
P = list(mean = c(1,1), cov = diag(1,2))
Q = list(mean = c(0,0), cov = 1.1*diag(1,2))
w2dist(P,Q)
```

wasserCostFunction *wasserCostFunction*

Description

Calculates the similarity distance matrix between elements j and i of a list of partitions.

Usage

```
wasserCostFunction(j, i, cytometries, equal.weights = FALSE)
```

Arguments

`j`	An entry of the list of partitions.
`i`	An entry of the list of partitions.
`cytometries`	The list of partitions.
`equal.weights`	If True, weights assigned to every cluster in a partion are uniform (1/number of clusters) when calculating the similarity distance. If False, weights assigned to clusters are the proportions of points in every cluster compared to the total amount of points in the partition.

Value

A similarity distance.

Examples

```
database = list(as.data.frame(Cytometry2)[which(match(Cytometry2$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry3)[which(match(Cytometry3$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry4)[which(match(Cytometry4$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry5)[which(match(Cytometry5$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry7)[which(match(Cytometry7$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry8)[which(match(Cytometry8$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry9)[which(match(Cytometry9$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry12)[which(match(Cytometry12$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry13)[which(match(Cytometry13$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry14)[which(match(Cytometry14$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry15)[which(match(Cytometry15$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry16)[which(match(Cytometry16$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry17)[which(match(Cytometry17$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry19)[which(match(Cytometry19$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),],
as.data.frame(Cytometry21)[which(match(Cytometry21$`Population ID (name)`,
c("Monocytes", "CD4+CD8-", "Mature SIg Kappa", "TCRgd-"), nomatch = 0)>0),])


templates.optimalFlow = optimalFlowTemplates(database = database, templates.number = 5,
cl.paral = 1)


print(wasserCostFunction(1,2,list(templates.optimalFlow$database.elliptical[[1]],
templates.optimalFlow$database.elliptical[[2]])))
```

`wasserMinDist` *wasserMinDist*

Description

For two lists of multivariate normals calcualtes the closest member, in wasserstein distance, of the later list to each element of the former.

Usage

```
wasserMinDist(points,centres)
```

Arguments

`points`	List of multivariate normals, where each element is a list with values mean and cov.
`centres`	List of multivariate normals, where each element is a list with values mean and cov.

Value

A matrix where each column idicates the distance between the respective entry in points and the closest element in centers and the index of this closest element.

Examples

```
normals = list(list(mean = c(1,1), cov = diag(2,2)),
list(mean = c(1,1), cov = diag(1,2)), list(mean = c(3,3), cov = diag(1,2)))
k_barycenter = kcenter(normals, 2, c(1,1,2))$kcenters
wasserMinDist(normals, k_barycenter)
```

`wassersteinKBarycenter`

wassersteinKBarycenter

Description

A wrapper for calculating K-barycenters of multivariate normal distributions with the 2-Wasserstein distance.

Usage

```
wassersteinKBarycenter(i = 1, k, alpha = 0, initialization = "rnd", pooled.clusters)
```

Arguments

`i`	A dummy variable ment for use with apply.
`k`	Number k of elements in the k-barycenter.
`alpha`	Level of trimming.
`initialization`	Type of initialization in c("rnd", "plus-plus"). "rnd" makes the common random initilaization while "plus-plus" initializes in a similar fashion to k-means++.

`pooled.clusters`
: List of multivariate normals for which the trimmed k-barycenter should be performed.

Value

`wasserstein.var`
: Wasserstein variation.

`wasserstein.k.barycenter`
: List with three elements. Variacion_wasser is Waserstein variation. Baricentro is a list of k elements, each of which is a member of the k-barycenter. Cluster is the assignation to each barycenter of the original entries.

References

E del Barrio, H Inouzhe, JM Loubes, C Matran and A Mayo-Iscar. (2019) optimalFlow: Optimal-transport approach to flow cytometry gating and population matching. arXiv:1907.08006

Examples

```
normals = list(list(mean = c(1,1), cov = diag(2,2)),
list(mean = c(1,1), cov = diag(1,2)), list(mean = c(3,3), cov = diag(1,2)))
wkb = wassersteinKBarycenter(1, 2, 0, "rnd", normals)
print(wkb$wasserstein.var)
print(wkb$wasserstein.k.barycente)
```

Index