Package 'DepecheR'

November 5, 2025

Version 1.27.0

Date 2024-08-22

Title Determination of essential phenotypic elements of clusters in high-dimensional entities

biocViews Software,CellBasedAssays,Transcription,DifferentialExpression, DataRepresentation,ImmunoOncology,Transcriptomics,Classification,Clustering, DimensionReduction,FeatureExtraction,FlowCytometry,RNASeq,SingleCell, Visualization

Description The purpose of this package is to identify traits in a dataset that can separate groups. This is done on two levels. First, clustering is performed, using an implementation of sparse K-means. Secondly, the generated clusters are used to predict outcomes of groups of individuals based on their distribution of observations in the different clusters. As certain clusters with separating information will be identified, and these clusters are defined by a sparse number of variables, this method can reduce the complexity of data, to only emphasize the data that actually matters.

License MIT + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 7.3.1

Depends R (>= 4.0)

Imports ggplot2 (>= 3.1.0), MASS (>= 7.3.51), Rcpp (>= 1.0.0), dplyr (>= 0.7.8), gplots (>= 3.0.1), viridis (>= 0.5.1), foreach (>= 1.4.4), doSNOW (>= 1.0.16), matrixStats (>= 0.54.0), mixOmics (>= 6.6.1), moments (>= 0.14), grDevices (>= 3.5.2), graphics (>= 3.5.2), stats (>= 3.5.2), utils (>= 3.5), methods (>= 3.5), parallel (>= 3.5.2), reshape2 (>= 1.4.3), beanplot (>= 1.2), FNN (>= 1.1.3), robustbase (>= 0.93.5), gmodels (>= 2.18.1), collapse (>= 1.9.2), ClusterR (>= 1.3.2)

LinkingTo Rcpp, RcppEigen

Suggests uwot, testthat, knitr, rmarkdown, BiocStyle

VignetteBuilder knitr

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

git_branch devel

git_last_commit eb6e467

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-05

Author Jakob Theorell [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8752-3151>),
Axel Theorell [aut]

Maintainer Jakob Theorell <jakob.theorell@ki.se>

Contents

DepecheR-package 2
dAllocate 3
dColorPlot 4
dColorVector 7
dContours 8
dDensityPlot 9
depeche 11
dResidualPlot 14
dScale 16
dSplsda 17
dViolins 20
dWilcox 22
groupProbPlot 25
microClust 27
neighSmooth 27
nUniqueNeighDons 29
testData 30
testDataDepeche 31
testDataSNE 31

Index **32**

`DepecheR-package` *DepecheR: Determination of essential phenotypic elements of clusters in high-dimensional entities*

Description

The purpose of this package is to identify traits in a dataset that can separate groups. This is done on two levels. First, clustering is performed, using an implementation of sparse K-means. Secondly, the generated clusters are used to predict outcomes of groups of individuals based on their distribution of observations in the different clusters. As certain clusters with separating information will be identified, and these clusters are defined by a sparse number of variables, this method can reduce the complexity of data, to only emphasize the data that actually matters.

Details

The package is indirectly clearly dependent on Rtsne for generation of output. See comment on the github wiki for how to speed Rtsne up.

Author(s)

Maintainer: Jakob Theorell <jakob.theorell@ki.se> (ORCID)

Authors:

- Axel Theorell <axel.theorell@gmail.com>

`dAllocate` *Allocation of observations to pre-established cluster centers.*

Description

Here, observations of a dataset are allocated to a set of preestablished cluster centers. This is intended to be used for the test set in train-test dataset situations.

Usage

```
dAllocate(inDataFrame, depModel)
```

Arguments

`inDataFrame`	A dataset that should be allocated to a set of cluster centers, for example a richer, but less representative dataset, with all datapoints from all donors, instead of only a set number of values from all.
`depModel`	This is the result of the original application of the depeche function on the associated, more representative dataset.

Value

A vector with the same length as number of rows in the inDataFrame, where the cluster identity of each observation is noted.

See Also

depeche

Examples

```
# Retrieve some example data
data(testData)
## Not run:
# Now arbitrarily (for the sake of the example) divide the data into a
# training- and a test set.
testDataSample <- sample(1:nrow(testData), size = 10000)
testDataTrain <- testData[testDataSample, ]
testDataTest <- testData[-testDataSample, ]

# Run the depeche function for the train set

depeche_train <- depeche(testDataTrain[, 2:15],
    maxIter = 20,
    sampleSize = 1000
)

# Allocate the test dataset to the centers of the train dataset
depeche_test <- dAllocate(testDataTest[, 2:15], depeche_train
)

# And finally plot the two groups to see how great the overlap was:
clustVecList <- list(list("Ids" =testDataTrain$ids,
                          "Clusters" = depeche_train$clusterVector),
                     list("Ids" =testDataTest$ids,
                          "Clusters" = depeche_test))
tablePerId <- do.call("rbind", lapply(seq_along(clustVecList), function(x){
                                      locDat <- clustVecList[[x]]
                                      locRes <- apply(as.matrix(table(
                                      locDat$Ids, locDat$Clusters)),
                                      1, function(y) y/sum(y))
                                      locResLong <- reshape2::melt(locRes)
                                      colnames(locResLong) <-
                                      c("Cluster", "Donor", "Fraction")
                                      locResLong$Group <- x
                                      locResLong
                                      }))
tablePerId$Cluster <- as.factor(tablePerId$Cluster)
tablePerId$Group <- as.factor(tablePerId$Group)

library(ggplot2)
ggplot(data=tablePerId, aes(x=Cluster, y=Fraction,
        fill=Group)) + geom_boxplot() + theme_bw()

## End(Not run)
```

---

dColorPlot *Display third variable as color on a 2D plot*

Description

Function to overlay one variable for a set of observations on a field created by two other variables known for the same observations. The plot is constructed primarily for displaying variables on 2D-stochastic neighbour embedding fields, but can be used for any sets of (two or) three variables known for the same observations. As the number of datapoints is often very high, the files would, if saved as pdf of another vector based file type become extremely big. For this reason, the plots are saved as jpeg and no axes or anything alike are added, to simplify usage in publications.

Usage

```
dColorPlot(
  colorData,
  controlData,
  xYData,
  colorScale = "rich_colors",
  plotName = "default",
  densContour = TRUE,
  title = FALSE,
  plotDir = "default",
  truncate = TRUE,
  bandColor = "black",
  dotSize = 500/sqrt(nrow(xYData)),
  continuous = "default",
  multiCore = "default",
  nCores = "default",
  createOutput = TRUE
)
```

Arguments

colorData	A numeric matrix or dataframe or a vector, be it numeric, charater or factor, that should be used to define the colors on the plot. A pre-made vector of colors is also accepted.
controlData	Optional. A numeric/integer vector or dataframe of values that could be used to define the range of the colorData. If no control data is present, the function defaults to using the colorData as control data.
xYData	These variables create the field on which the colorData will be displayed. It needs to be a matrix or dataframe with two columns and the same number of rows as the colorData object.
colorScale	This argument controls the colors in the plot. See dColorVector for alternatives.
plotName	The name(s) for the plot(s). 'default' returns the column names of the colorData object in the case this is a dataframe and otherwise returns the somewhat generic name 'testVariable'. It can be substituted with a string (in the case colorData is a vector) or vector of strings, as long as it has the same length as the number of columns in colorData.

densContour	If density contours should be created for the plot(s) or not. Defaults to TRUE. If a density object, as generated by dContours, is included, this will be used instead.
title	If there should be a title displayed on the plotting field. As the plotting field is saved a jpeg, this title cannot be removed as an object afterwards, as it is saved as coloured pixels. To simplify usage for publication, the default is FALSE, as the files are still named, eventhough no title appears on the plot.
plotDir	If different from the current directory. If specified and non-existent, the function creates it. If "." is specified, the plots will be saved at the current directory. By default, a new directory is added if the created plots will be more than 1.
truncate	If truncation of the most extreme values should be performed for the visualizations. Three possible values: TRUE, FALSE, and a vector with two values indicating the low and high threshold quantiles for truncation.
bandColor	The color of the contour bands. Defaults to black.
dotSize	Simply the size of the dots. The default makes the dots maller the more observations that are included.
continuous	Boolean. Is the colorData parameter continuous? If default, then only numeric vectors with more than 20 values are considered continuous.This only applies to situations with single vectors. In situations where a dataframe is added as colorData, all variables are considered continuous.
multiCore	If the algorithm should be performed on multiple cores. This increases the speed if the dataset is medium-large (>100000 rows) and has at least 5 columns. Default is TRUE when these above criteria are met and FALSE otherwise.
nCores	If multiCore is TRUE, then this sets the number of parallel processes. The default is currently 87.5 percent with a cap on 10 cores, as no speed increase is generally seen above 10 cores for normal computers.
createOutput	For testing purposes. Defaults to TRUE. If FALSE, no plots are generated.

Value

Plots showing the colorData displayed as color on the field created by xYData.

See Also

dDensityPlot, dResidualPlot, dWilcox, dColorVector

Examples

```
# Load some data
data(testData)
## Not run:
# Load or create the dimensions that you want to plot the result over.
# uwot::umap recommended due to speed, but tSNE or other method would
# work as fine.
data(testDataSNE)

# Run the function for two of the variables
```

```
dColorPlot(colorData = testData[2:3], xYData = testDataSNE$Y)

# Now each depeche cluster is plotted separately and together.

# Run the clustering function. For more rapid example execution,
# a depeche clustering of the data is included
# testDataDepeche <- depeche(testData[,2:15])
data(testDataDepeche)

dColorPlot(
    colorData = testDataDepeche$clusterVector,
    xYData = testDataSNE$Y, plotName = "clusters"
)

## End(Not run)
```

---

dColorVector *Create a vector of colors of the same length as the data*

Description

This function takes a vector x and a shorter ordering vector with all the unique values of the x vector in the specific order that the colors should be in and returns a vector of RGB colors the same length as the initial x vector.

Usage

```
dColorVector(x, colorOrder = unique(x), colorScale = "viridis")
```

Arguments

x	Any vector.
colorOrder	The order that the colors should be in in the output vector. Defaults to the order that the unique values in x occurs.
colorScale	The color scale. Inherited from the viridis, gplots and grDevices packages (and the package-specific 'dark_rainbow'). Seven possible scales are pre-made: inferno, magma, plasma, viridis, rich_colors, rainbow and dark_rainbow. User specified vectors of colors (e.g. c('#FF0033', '#03AF49')) are also accepted.

Value

A vector, the same length as x with each unique value substitutet with a color.

See Also

dDensityPlot, dColorPlot, dViolins

Examples

```
# Load some data
data(testData)

testColor <- dColorVector(testData$ids, colorScale = "plasma")

# In this case, each of the 97 individual donors in the dataset has gotten
# their own color code:
table(testColor)
```

---

dContours *Create density contours for two-dimensional data.*

Description

Here, contour lines for two-dimensional data are construced. It is primarily thought to be used in the context of SNE plots in this package. This function is used both internally in other functions suchas sneFluoroPlot and sneDensityPlot, but also as a standalone function, as it increases speed greatly to generate the density curves only once per overall analysis.

Usage

```
dContours(xYData, control, n = 100)
```

Arguments

xYData	A dataframe with two columns containing position information for each observation in the dataset. Typically, this is the raw result from the SNE analysis.
control	A numeric/integer vector or dataframe of values that could be used to define the range in the internal dScale. If no control data is present, the function defaults to using the indata as control data.
n	The number fo grid points. Default is 100.

Value

A list of three components

x, y The x and y coordinates of the grid points, vectors of length n.

z An n[1] by n[2] matrix of the estimated density: rows correspond to the value of x, columns to the value of y.

See Also

dColorPlot, dDensityPlot, dResidualPlot, dWilcox

Examples

```
# Load the test SNE data
data(testDataSNE)


# Run the function
contour_result <- dContours(testDataSNE$Y)
```

---

dDensityPlot *Display density on 2D plot*

Description

Function to show density for a set of observations on a field created by two variables. The plot is constructed primarily for displaying density of 2D-stochastic neighbour embedding fields, but can be used for any sets of two known for the same observations. As the number of datapoints is often very high, the files would, if saved as pdf of another vector based file type become big. For this reason, the plots are saved as jpeg and no axes or anything alike are added, to simplify usage in publications.

Usage

```
dDensityPlot(
  xYData,
  colorScale = "default",
  plotName = "All_density",
  idsVector,
  densContour = TRUE,
  title = FALSE,
  plotDir = "default",
  bandColor = "black",
  dotSize = 500/sqrt(nrow(xYData)),
  createOutput = TRUE
)
```

Arguments

xYData	A dataframe or matrix with two columns. Each row contains information about the x and y positition in the field for that observation.
colorScale	This gives the specific color for the densest part of the plot(s). It has three possible values: **A specific color, e.g. 'red' or '#FF0000'** If no idsVector provided **A color scale from dColorVector** If idsVector provided. See dColorVector for alternatives. **"default"** "One color (blue) if no idsVector is provided, and otherwise the viridis color scale.

plotName	A name that is common to all density plots created. It can be the groups name, e.g. 'Malaria patients' or 'Clusters'. If only one plot is created, the name is still taken from here.
idsVector	Optional. Vector with the same length as xYData containing information about the id of each observation. If provided, density plots for each individual id and all ids together are produced.
densContour	If density contours should be created for the plot(s) or not. Defaults to TRUE. If a density object, as generated by dContours, is included, this will be used for the internal scaling of the plot, allowing for density distribution checks of different subcompartments of the data with the same scaling.
title	If there should be a title displayed on the plotting field. As the plotting field is saved as a png, this title cannot be removed as an object afterwards, as it is saved as coloured pixels. To simplify usage for publication, the default is FALSE, as the files are still named, eventhough no title appears on the plot.
plotDir	If different from the current directory. If not "." and non-existent, the function creates it. Default is "." if idsVector is not specified and otherwise paste0("Density plots for ", plotName, "s").
bandColor	The color of the contour bands. Defaults to black.
dotSize	Simply the size of the dots. The default makes the dots smaller the more observations that are included.
createOutput	For testing purposes. Defaults to TRUE. If FALSE, no output is generated.

Value

Plots showing the densities of the specific xYData (subset) displayed as color on the field created by the same xYData (subset).

See Also

dColorPlot, dResidualPlot, dWilcox, dColorVector

Examples

```
# Load some data
data(testData)
## Not run:
# Load or create the dimensions that you want to plot the result over.
# uwot::umap recommended due to speed, but tSNE or other method would
# work as fine.
data(testDataSNE)

# Plot all data together
dDensityPlot(xYData = testDataSNE$Y)

# Now each depeche cluster is plotted separately and together.

# Run the clustering function. For more rapid example execution,
# a depeche clustering of the data is included
```

```
# testDataDepeche <- depeche(testData[,2:15])
data(testDataDepeche)

dDensityPlot(
    xYData = testDataSNE$Y,
    idsVector = testDataDepeche$clusterVector,
    plotName = "cluster"
)

## End(Not run)
```

---

depeche *Perform optimization and penalized K-means clustering*

Description

This is the central function of the package. As input, only a dataset is required. It starts by performing optimizations and then performs clustering based on the values identified in the optimization step.

Usage

```
depeche(
  inDataFrame,
  samplingSubset = seq_len(nrow(inDataFrame)),
  penalties = 2^seq(0, 5, by = 0.5),
  sampleSize = "default",
  selectionSampleSize = "default",
  k = 30,
  minARIImprovement = 0.01,
  optimARI = 0.95,
  maxIter = 100,
  log2Off = FALSE,
  center = "default",
  scale = TRUE,
  nCores = "default",
  plotDir = ".",
  createOutput = TRUE
)
```

Arguments

inDataFrame	A dataframe or matrix with the data that will be used to create the clustering. Cytometry data should be transformed using biexponential, arcsinh transformation or similar, and day-to-day normalizations should to be performed for all data if not all data has been acquired on the same run. Scaling, etc, is on the other hand performed within the function.

`samplingSubset` If the dataset is made up of an unequal number of cells from multiple individuals, it might be wise to pre-define a subset of the rows, which includes equal or near-equal numbers of cells from each individual, to avoid a few outliers to dominate the analysis. This can be done here. Should be a vector of row numbers in the inDataFrame.

`penalties` This argument decides whether a single penalty will be used for clustering, or if multiple penalties will be evaluated to identify the optimal one. A single value, a vector of values, or possibly a list of two vectors, if dual clustering is performed can be given here. The suggested default values are empirically defined and might not be optimal for a specific dataset, but the algorithm will warn if the most optimal values are on the borders of the range. Note that when the penalty is 0, there is no penalization, which means that the algorithm runs standard K-means clustering.

`sampleSize` This controls what fraction of the dataset that will be used to run the penalty optimization. 'default' results in the full file in files up to 10000 events. In cases where the sampleSize argument is larger than 10000, default leads to the generation of a random subset to the same size also for the selectionSampleSize. A user specified number is also accepted.

`selectionSampleSize`
The size of the dataset used to find the optimal solution out of the many generated by the penalty optimization at each sample size. 'default' results in the full file in files up to 10000 events. In cases where the sampleSize argument is larger than 10000, default leads to the generation of a random subset to the same size also for the selectionSampleSize. A user specified number is also accepted.

`k` Number of initial cluster centers. The higher the number, the greater the precision of the clustering, but the computing time also increases linearly with the number of starting points. Default is 30. If penalties=0, k-means clustering with k clusters will be performed.

`minARIImprovement`
This is the stop criterion for the penalty optimization algorithm: the more iterations that are run, the smaller will the improvement of the corrected Rand index be, and this sets the threshold when the inner iterations stop. Defaults to 0.01.

`optimARI` Above this level of ARI, all solutions are considered equally valid, and the median solution is selected among them.

`maxIter` The maximal number of iterations that are performed in the penalty optimization.

`log2Off` If the automatic detection for high kurtosis, and followingly, the log2 transformation, should be turned off.

`center` If centering should be performed. Alternatives are 'default', 'mean', 'peak', FALSE and a vector of numbers with the same length as the number of columns in the inDataFrame. 'peak' results in centering around the highest peak in the data, which is useful in most cytometry situations. 'mean' results in mean centering. 'default' gives different results depending on the data: datasets with 100+ variables are mean centered, and otherwise, peak centering is used. If a numeric vector is provided, it is used to center the values to the numbers. This is preferable to pre-centering the data and using the FALSE command, as it will lead

to better internal visualization procedures, etc. FALSE results in no centering, mainly for testing purposes.

`scale` If scaling should be performed. If TRUE, the dataset will be divided by the combined standard deviation of the whole dataset. If a number is provided, the dataset is divided by this number. This scaling procedure makes the default penalties fit most datasets with some precision.

`nCores` If multiCore is TRUE, then this sets the number of parallel processes. The default is currently 87.5 percent with a cap on 10 cores, as no speed increase is generally seen above 10 cores for normal computers.

`plotDir` Where should the diagnostic plots be printed?

`createOutput` For testing purposes. Defaults to TRUE. If FALSE, no plots are generated.

Value

A nested list:

clusterVector A vector with the same length as number of rows in the inDataFrame, where the cluster identity of each observation is noted.

clusterCenters A matrix containing information about where the centers are in all the variables that contributed to creating the cluster with the given penalty term. An exact zero here indicates that the variable in question was sparsed out for that cluster. If a variable did not contribute to the separation of any cluster, it will not be present here.

essenceElementList A per-cluster list of the items that were used to separate that cluster from the rest, i.e. the items that survived the penalty.

penaltyOptList A list of two dataframes:

penaltyOpt.df A one row dataframe with the settings for the optimal penalty.

meanOptimDf A dataframe with the information about the results with all tested penalty values.

logCenterScale The values used to center and scale the data and information on if the data was log transformed. This information is used internally in dAllocate.

Examples

```
# Load some data
data(testData)

# Here a run with the standard settings
## Not run:
testDataDepecheResult <- depeche(testData[, 2:15])

# Look at the result
str(testDataDepecheResult)

## End(Not run)
```

`dResidualPlot` *Showing the residuals when subtracting the values from one group from another on a SNE plot*

Description

This function is used to visually compare groups of individuals from whom comparable cytometry or other complex data has been generated, but where the number of individuals does not permit any statistical comparisons.

Usage

```
dResidualPlot(
  xYData,
  groupVector,
  clusterVector,
  densContour = TRUE,
  groupName1 = unique(groupVector)[1],
  groupName2 = unique(groupVector)[2],
  plotName = "default",
  title = FALSE,
  maxAbsPlottingValues,
  bandColor = "black",
  plotDir = ".",
  dotSize = 400/sqrt(nrow(xYData)),
  createOutput = TRUE
)
```

Arguments

`xYData`	A dataframe or matrix with two columns. Each row contains information about the x and y positition in the field for that observation.
`groupVector`	Vector with the same length as xYData containing information about the group identity of each observation.
`clusterVector`	Vector with the same length as xYData containing information about the cluster identity of each observation.
`densContour`	If density contours should be created for the plot(s) or not. Defaults to TRUE.
`groupName1`	The name for the first group
`groupName2`	The name for the second group
`plotName`	The main name for the graph and the analysis.
`title`	If there should be a title displayed on the plotting field. As the plotting field is saved as a png, this title cannot be removed as an object afterwards, as it is saved as coloured pixels. To simplify usage for publication, the default is FALSE, as the files are still named, eventhough no title appears on the plot.

maxAbsPlottingValues
: If multiple plots should be compared, it might be useful to define a similar color scale for all plots, so that the same color always means the same value. Such a value can be added here. It defaults to the maximum Wilcoxon statistic that is generated in the analysis.

bandColor
: The color of the contour bands. Defaults to black.

plotDir
: If different from the current directory. If specified and non-existent, the function creates it. If "." is specified, the plots will be saved at the current directory.

dotSize
: Simply the size of the dots. The default makes the dots smaller the more observations that are included.

createOutput
: For testing purposes. Defaults to TRUE. If FALSE, no plots are generated.

Value

A sne based plot showing which events that belong to a cluster dominated by the first or the second group.

See Also

dColorPlot, dDensityPlot, dWilcox

Examples

```
# Load some data
data(testData)
## Not run:
# Load or create the dimensions that you want to plot the result over.
# uwot::umap recommended due to speed, but tSNE or other method would
# work as fine.
data(testDataSNE)

# Run the clustering function. For more rapid example execution,
# a depeche clustering of the data is inluded
# testDataDepeche <- depeche(testData[,2:15])
data(testDataDepeche)


# And finally run the function
dResidualPlot(
    xYData = testDataSNE$Y, groupVector = testData[, 16],
    clusterVector = testDataDepeche$clusterVector
)

## End(Not run)
```

dScale *Scaling of a vector or a dataframe.*

Description

This is a scaling function with a number of alternatives. This method for scaling takes the shape of the data into somewhat more of a consideration than minMaxScale does, but still gives less influence of outliers than more conventional scalin alternatives, such as unit variance scaling.

Usage

```
dScale(
  x,
  control,
  scale = TRUE,
  robustVarScale = TRUE,
  center = "peak",
  truncate = FALSE,
  multiplicationFactor = 1,
  returnCenter = FALSE,
  nCores = "default"
)
```

Arguments

x	A numeric/integer vector or dataframe
control	A numeric/integer vector or dataframe of values that could be used to define the range. If no control data is present, the function defaults to using the indata as control data.
scale	If scaling should be performed. Three possible values: a vector with two values indicating the low and high threshold quantiles for the scaling, TRUE, which equals the vector 'c(0.001, 0.999)', and FALSE.
robustVarScale	If the data should be scaled to its standard deviation within the quantiles defined by the scale values above. If TRUE (the default), the data is unit variance scaled based on the standard deviation of the data within the range defined by scale.
center	If centering should be performed. Alternatives are mean', 'peak' and FALSE. 'peak' results in centering around the highest peak in the data, which is useful in most cytometry situations. 'mean' results in mean centering.
truncate	If truncation of the most extreme values should be performed. Three possible values: TRUE, FALSE, and a vector with two values indicating the low and high threshold quantiles for truncation.
multiplicationFactor	A value that all values will be multiplied with. Useful e.g. if the results preferrably should be returned as percent.
returnCenter	Boolean. If center=TRUE, should the value at the center be returned?

nCores — If the function is run in multicore mode, which it will if the dataset is large (nrow*ncol>10^6), this decides the number of cores. The default is currently 87.5 percent with a cap on 10 cores, as no speed increase is generally seen above 10 cores for normal computers to date.

Value

A vector or dataframe with the same size but where all values in the vector or column of the dataframe have been internally scaled. In addition, if returnCenter=TRUE, a value, or a vector if x is a matrix or a data frame.

Examples

```
# Load some data
data(testData)

# Retrieve the first column
x <- testData[, 2]

# The maximum and minimum values are
max(x)
min(x)

# Run the function without mean centering and with the quantiles set to 0
# and 1.
y <- dScale(x, scale = c(0, 1), robustVarScale = FALSE, center = FALSE)

# And the data has been scaled to the range between 0 and 1.
max(y)
min(y)

# Now run the default function for a dataframe
summary(testData[, 2:15])

y_df <- dScale(testData[, 2:15])

# Here, the data has first been truncated to the default percentiles, then
# scaled to the standard deviation in the remaining interval and finally the
# center has been placed where the highest peak in the data is present.
# NB! Here, no truncation has been performed in the scaling, only to obtain
# the scaling values.

summary(y_df)
```

---

dSplsda — *Sparse partial least squares discriminant analysis with paired and unpaired data*

Description

This function is used to compare groups of individuals from whom comparable cytometry or other complex data has been generated. It is superior to just running a Wilcoxon analysis in that it does not consider each cluster individually, but instead uses a sparse partial least squares discriminant analysis to first identify which vector thourgh the multidimensional data cloud, created by the cluster-donor matrix, that optimally separates the groups, and as it is a sparse algorithm, applies a penalty to exclude the clusters that are orthogonal, or almost orthogonal to the discriminant vector, i.e. that do not contribute to separating the groups. This is in large a wrapper for the `splsda` function from the mixOmics package.

Usage

```
dSplsda(
  xYData,
  idsVector,
  groupVector,
  clusterVector,
  displayVector,
  testSampleRows,
  paired = FALSE,
  densContour = TRUE,
  plotName = "default",
  groupName1 = unique(groupVector)[1],
  groupName2 = unique(groupVector)[2],
  thresholdMisclassRate = 0.05,
  title = FALSE,
  plotDir = ".",
  bandColor = "black",
  dotSize = 500/sqrt(nrow(xYData)),
  createOutput = TRUE
)
```

Arguments

`xYData`	A dataframe or matrix with two columns. Each row contains information about the x and y positition in the field for that observation.
`idsVector`	Vector with the same length as xYData containing information about the id of each observation.
`groupVector`	Vector with the same length as xYData containing information about the group identity of each observation.
`clusterVector`	Vector with the same length as xYData containing information about the cluster identity of each observation.
`displayVector`	Optionally, if the dataset is very large (>100 000 observations) and hence the SNE calculation becomes impossible to perform for the full dataset, this vector can be included. It should contain the set of rows from the data used for statistics, that has been used to generate the xYData.

`testSampleRows` Optionally, if a train-test setup is wanted, the rows specified in this vector are used to divide the dataset into a training set, used to generate the analysis, and a test set, where the outcome is predicted based on the outcome of the training set. All rows that are not labeled as test rows are assumed to be train rows.

`paired` Defaults to FALSE, i.e. no assumption of pairing is made and Wilcoxon rank sum-test is performed. If true, the software will by default pair the first id in the first group with the firs id in the second group and so forth, so make sure the order is correct!

`densContour` If density contours should be created for the plot(s) or not. Defaults to TRUE. a

`plotName` The main name for the graph and the analysis.

`groupName1` The name for the first group

`groupName2` The name for the second group

`thresholdMisclassRate` This threshold corresponds to the usefulness of the model in separating the groups: a misclassification rate of the default 0.05 means that 5 percent of the individuals are on the wrong side of the theoretical robust middle line between the groups along the sPLS-DA axis, defined as the middle point between the 3:rd quartile of the lower group and the 1:st quartile of the higher group.

`title` If there should be a title displayed on the plotting field. As the plotting field is saved as a png, this title cannot be removed as an object afterwards, as it is saved as coloured pixels. To simplify usage for publication, the default is FALSE, as the files are still named, eventhough no title appears on the plot.

`plotDir` If different from the current directory. If specified and non-existent, the function creates it. If "." is specified, the plots will be saved at the current directory.

`bandColor` The color of the contour bands. Defaults to black.

`dotSize` Simply the size of the dots. The default makes the dots smaller the more observations that are included.

`createOutput` For testing purposes. Defaults to TRUE. If FALSE, no output is generated.

Value

This function returns the full result of the sPLS-DA. It also returns a SNE based plot showing which events that belong to a cluster dominated by the first or the second group defined by the sparse partial least squares loadings of the clusters.

See Also

splsda, dColorPlot, dDensityPlot, dResidualPlot

Examples

```
# Load some data
data(testData)
## Not run:
# Load or create the dimensions that you want to plot the result over.
# uwot::umap recommended due to speed, but tSNE or other method would
```

```
# work as fine.
data(testDataSNE)

# Run the clustering function. For more rapid example execution,
# a depeche clustering of the data is inluded
# testDataDepeche <- depeche(testData[,2:15])
data(testDataDepeche)


# Run the function. This time without pairing.
sPLSDAObject <- dSplsda(
    xYData = testDataSNE$Y, idsVector = testData$ids,
    groupVector = testData$label,
    clusterVector = testDataDepeche$clusterVector
)


# Here is an example of how the display vector can be used.
subsetVector <- sample(1:nrow(testData), size = 10000)

# Now, the SNE for this displayVector could be created
# testDataSubset <- testData[subsetVector, 2:15]
# testDataSNESubset <- Rtsne(testDataDisplay, pca=FALSE)$Y
# But we will just subset the testDataSNE immediately
testDataSNESubset <- testDataSNE$Y[subsetVector, ]

# And now, this new SNE can be used for display, although all
# the data is used for the sPLS-DA calculations
sPLSDAObject <- dSplsda(
    xYData = testDataSNESubset, idsVector = testData$ids,
    groupVector = testData$label, clusterVector =
        testDataDepeche$clusterVector,
    displayVector = subsetVector
)

# Finally, an example of a train-test set situation, where a random half the
# dataset is used for training and the second half is used for testing. It
# is naturally more biologically interesting to use two independent datasets
# for training and testing in the real world.
sPLSDAObject <- dSplsda(
    xYData = testDataSNE$Y, idsVector = testData$ids,
    groupVector = testData$label, clusterVector =
        testDataDepeche$clusterVector, testSampleRows = subsetVector
)

## End(Not run)
```

---

dViolins *Create violin plots for any variables of choise*

Description

Here, assymetrical violin plots for each cluster vs all other clusters are plotted for variables either retrieved from a depeche analysis or user-defined.

Usage

```
dViolins(
  clusterVector,
  inDataFrame,
  plotClusters = unique(clusterVector),
  plotElements = "all",
  colorOrder = plotClusters,
  colorScale = "viridis",
  plotDir = "dViolin_result",
  createOutput = TRUE
)
```

Arguments

clusterVector	Vector with the same length as inDataFrame containing information about the cluster identity of each observation.
inDataFrame	The data used to generate the depecheObject
plotClusters	This vector of numbers define which cluster(s) to plot the violins for. Defaults to all.
plotElements	This provides information on which features to plot. In the typical case, this is the essenceElementList from a depeche run. Other input formats are however accepted: if a vector of column names is provided, then these features will be plotted for all clusters. A custom list of features specific for each cluster is also accepted. A final alternative is to return "all" (default), in which case all markers will be plotted for all clusters.If more than a 100 markers are provided, however, this will return an error.
colorOrder	The order of the cluster colors. Defaults to the order that the unique values in clusterVector occurs.
colorScale	The color scale. Options identical to dColorVector.
plotDir	The name of the created directory.
createOutput	For testing purposes. Defaults to TRUE. If FALSE, no plots are generated.

Value

One graph is created for each cluster, containing a bean per specified variable.

See Also

dDensityPlot, dColorPlot, dColorVector, depeche

Examples

```
# Load some data
data(testData)

# Run the clustering function. For more rapid example execution,
# a depeche clustering of the data is inluded
# testDataDepeche <- depeche(testData[,2:15])
data(testDataDepeche)

# Create the plots of the variables that contribute to creating cluster 3

## Not run:
dViolins(testDataDepeche$clusterVector,
    inDataFrame = testData,
    plotClusters = 3, plotElements = testDataDepeche$essenceElementList
)

## End(Not run)
```

---

dWilcox *Wilcoxon rank-sum or signed rank test comparison of subject groups in a dClust result*

Description

This function is used to compare groups of individuals from whom comparable cytometry or other complex data has been generated.

Usage

```
dWilcox(
  xYData,
  idsVector,
  groupVector,
  clusterVector,
  displayVector,
  paired = FALSE,
  multipleCorrMethod = "BH",
  densContour = TRUE,
  plotName = "default",
  groupName1 = unique(groupVector)[1],
  groupName2 = unique(groupVector)[2],
  title = FALSE,
  lowestPlottedP = 0.05,
  plotDir = ".",
  bandColor = "black",
  dotSize = 500/sqrt(nrow(xYData)),
  createOutput = TRUE
)
```

Arguments

xYData	A dataframe or matrix with two columns. Each row contains information about the x and y positition in the field for that observation.
idsVector	Vector with the same length as xYData containing information about the id of each observation.
groupVector	Vector with the same length as xYData containing information about the group identity of each observation.
clusterVector	Vector with the same length as xYData containing information about the cluster identity of each observation.
displayVector	Optionally, if the dataset is very large and the SNE calculation hence becomes impossible to perform for the full dataset, this vector can be included. It should contain the set of rows from the data used for statistics, that has been used to generate the xYData.
paired	Defaults to FALSE, i.e. no assumption of pairing is made and Wilcoxon rank sum-test is performed. If true, the software will by default pair the first id in the first group with the firs id in hte second group and so forth.
multipleCorrMethod	Which method that should be used for adjustment of multiple comparisons. Defaults to Benjamini-Hochberg, but all other methods available in p.adjust can be used.
densContour	If density contours should be created for the plot(s) or not. Defaults to TRUE. a
plotName	The main name for the graph and the analysis.
groupName1	The name for the first group
groupName2	The name for the second group
title	If there should be a title displayed on the plotting field. As the plotting field is saved as a png, this title cannot be removed as an object afterwards, as it is saved as coloured pixels. To simplify usage for publication, the default is FALSE, as the files are still named, eventhough no title appears on the plot.
lowestPlottedP	If multiple plots should be compared, it might be useful to define a similar color scale for all plots, so that the same color always means the same statistical value. A p-value that determines this can be added here. Default is a p-value of 0.05. In cases where no datapoints have any lower p-values than this, a Wilcoxon-statistic corresponding as closely as possible to 0.05 will be identified with iterations of datasets with the same size as indicated by hte group vector. If one value is lowerthan 0.05, the wilcoxon statistic from this comparison is used instead.
plotDir	If different from the current directory. If specified and non-existent, the function creates it. If "." is specified, the plots will be saved at the current directory.
bandColor	The color of the contour bands. Defaults to black.
dotSize	Simply the size of the dots. The default makes the dots smaller the more observations that are included.
createOutput	For testing purposes. Defaults to TRUE. If FALSE, no plots are generated.

Value

This function always returns a dataframe showing the Wilcoxon statistic and the p-value for each cluster, with an included adjustment for multiple comparisons (see above). It also returns a sne based plot showing which events that belong to a cluster dominated by the first or the second group.

See Also

dColorPlot, dDensityPlot, dResidualPlot

Examples

```
# Load some data
data(testData)
## Not run:
# Load or create the dimensions that you want to plot the result over.
# uwot::umap recommended due to speed, but tSNE or other method would
# work as fine.
data(testDataSNE)

# Run the clustering function. For more rapid example execution,
# a depeche clustering of the data is inluded
# testDataDepeche <- depeche(testData[,2:15])
data(testDataDepeche)

# Run the function
dWilcoxResult <- dWilcox(
    xYData = testDataSNE$Y, idsVector = testData$ids,
    groupVector = testData$label, clusterVector =
    testDataDepeche$clusterVector
)

# Here is an example of how the display vector can be used.
subsetVector <- sample(1:nrow(testData), size = 10000)

# Now, the SNE for this displayVector could be created
# testDataSubset <- testData[subsetVector, 2:15]
# testDataSNESubset <- Rtsne(testDataDisplay, pca=FALSE)$Y
# But we will just subset the testDataSNE immediately
testDataSNESubset <- testDataSNE$Y[subsetVector, ]

# And now, this new SNE can be used for display, although all
# the data is used for the Wilcoxon calculations
dWilcoxResult <- dWilcox(
    xYData = testDataSNESubset, idsVector = testData$ids,
    groupVector = testData$label, clusterVector =
        testDataDepeche$clusterVector, displayVector = subsetVector
)

## End(Not run)
```

groupProbPlot *Define and plot group probabilities*

Description

This function defines and plots the single-observation probability for belonging to either of two groups. It uses the neighSmooth function with the special case that the values are binary: For each set of k nearest neighbors, cell x is assigned a probability to belong to one group or the other based on the percentage of the neighbors belonging to each group. In other words, if 20 out of 100 neighbors belong to group A and 80 belong to group B, and the value for the cell will be 20 A or 80 accordingly reflected in the color scale on the resulting plot.

Usage

```
groupProbPlot(
  xYData,
  groupVector,
  euclidSpaceData,
  kNeighK = max(100, round(nrow(euclidSpaceData)/10000)),
  kMeansK = round(nrow(euclidSpaceData)/1000),
  densContour = TRUE,
  groupName1 = unique(groupVector)[1],
  groupName2 = unique(groupVector)[2],
  plotName = "default",
  title = FALSE,
  bandColor = "black",
  plotDir = ".",
  dotSize = 400/sqrt(nrow(xYData)),
  returnProb = FALSE,
  returnProbColVec = FALSE,
  createOutput = TRUE
)
```

Arguments

xYData	A dataframe or matrix with two columns. Each row contains information about the x and y positition in the field for that observation.
groupVector	Vector with the same length as xYData containing information about the group identity of each observation.
euclidSpaceData	The data cloud in which the nearest neighbors for the events should be identified.
kNeighK	The number of nearest neighbors.
kMeansK	The number of clusters in the initial step of the algorithm. A higher number leads to shorter runtime, but potentially lower accuracy.
densContour	If density contours should be created for the plot(s) or not. Defaults to TRUE. a

groupName1	The name for the first group
groupName2	The name for the second group
plotName	The main name for the graph and the analysis.
title	If there should be a title displayed on the plotting field. As the plotting field is saved as a png, this title cannot be removed as an object afterwards, as it is saved as coloured pixels. To simplify usage for publication, the default is FALSE, as the files are still named, eventhough no title appears on the plot.
bandColor	The color of the contour bands. Defaults to black.
plotDir	If different from the current directory. If specified and non-existent, the function creates it. If "." is specified, the plots will be saved at the current directory.
dotSize	Simply the size of the dots. The default makes the dots smaller the more observations that are included.
returnProb	Should a probability vector be returned? Mutually exclusive with returnProbColVec.
returnProbColVec	Should the color vector be returned as part of the output? Mutually exclusive with returnProb.
createOutput	For testing purposes. Defaults to TRUE. If FALSE, no output is generated.

Value

A graph showing the probability as a color scale from blue over white to red for each event to belong to one group or the other, with a separate color scale. Optionally also the color vector, if returnProbColVec is TRUE.

Examples

```
data(testData)
data(testDataSNE)
euclidSpaceData <-
    testData[, c(
        "SYK", "CD16", "CD57", "EAT.2",
        "CD8", "NKG2C", "CD2", "CD56"
    )]
## Not run:
groupProbPlot(
    xYData = testDataSNE$Y, groupVector = testData$label,
    euclidSpaceData
)

## End(Not run)
```

microClust — *This function is the core of the neighSmooth. See the documentation there for details.*

Description

This function is the core of the neighSmooth. See the documentation there for details.

Usage

```
microClust(
  dataCenter,
  dataNeigh,
  dataReturn,
  method = "median",
  k = 11,
  trim = 0
)
```

Arguments

dataCenter	The original data.
dataNeigh	The data for the neighbors. Often stronly overlapping with the dataCenter, but for internal reasons, this data cloud is larger than the dataCenter cloud.
dataReturn	The neighbor data that should be aggregated and sent back.
method	Should median or mean be calculated?
k	Number of neighbors.
trim	If mean of the neighbors is returned, should it be calculated with trimming?

Value

A dataset with the same shape as dataCenter, filled with aggregated information from the k nearest neighbors.

neighSmooth — *Euclidean neighbor smoothing*

Description

This function constructs a variable that for each event shows the average value for its euclidean k-nearest neighbors. It builds on the same idea as has been put forward in the Sconify package: -Burns TJ (2019). Sconify: A toolkit for performing KNN-based statistics for flow and mass cytometry data. R package version 1.4.0 and -Hart GT, Tran TM, Theorell J, Schlums H, Arora G, Rajagopalan S, et al. Adaptive NK cells in people exposed to Plasmodium falciparum correlate with protection from malaria. J Exp Med. 2019 Jun 3;216(6):1280–90. First, the k nearest neighbors are defined for cell x. Then, the average value for the k nearest neighbors is returned as the result for cell x.

Usage

```
neighSmooth(
  focusData,
  euclidSpaceData,
  neighRows = "default",
  ctrlRows = NULL,
  kNeighK = "default",
  kMeansK = "default",
  kMeansCenters = NULL,
  kMeansClusters = NULL,
  method = "mean",
  nCores = detectCores() - 1
)
```

Arguments

focusData	The data that should be smoothed. Should be a matrix with the variables to be smoothed as columns.
euclidSpaceData	The data cloud in which the nearest neighbors for the events should be identified. Can be a vector, matrix or dataframe. It is worth noting that if this data has more than 10 dimensions, the first step of the algorithm will be the creation of a 10-dimensional PCA using fast.prcomp from gmodels. So in cases where this function is used iteratively, it might be wiser to run the PCA beforehand.
neighRows	The rows in the dataset that correspond to the neighbors of the focusData points. "default" is all the focusData points, but a subset can be added instead, if preferred. This is good to use to increase robustness, e.g. by running 100 iterations with different sets of neighbors with the same number of points from each group/individual.
ctrlRows	Optionally, a set of control rows that are used to remove background signal from the neighRows data before sending the data back.
kNeighK	The number of nearest neighbors. "default" is the max of 100 and the number of neighbor rows divided by 10000. Mutliple different values here is preferred.
kMeansK	The number of clusters in the initial step of the algorithm. A higher number leads to shorter runtime, but potentially lower accuracy. This is not used if kMeansCenters is provided. "default" is the highest of 1 and the number of cells in euclidSpaceData divided by 1000.
kMeansCenters	Here, a pre-clustering of the data can be provided, in which case the clustering will not be performed internally. Wise if for example a bootstrapping scheme is used to define the neighRows iteratively, as the k-means step can be quite time consuming. This part is the cluster centers or centroids.
kMeansClusters	See above. Here, the clusters, instead of the centroids are provided if used.
method	The method to use for the smoothing. Three values possible: mean (default), median and mode.
nCores	The number of cores used. Defaults to number of cores in the computer minus 1.

Value

An object of the same dimensions as focusData that has been smoothed.

Examples

```
data(testData)
data(testDataSNE)
euclidSpaceData <-
    testData[, c(
        "SYK", "CD16", "CD57", "EAT.2",
        "CD8", "NKG2C", "CD2", "CD56"
    )]
## Not run:
smoothGroupVector <- neighSmooth(
    focusData = as.numeric(testData$label),
    euclidSpaceData
)

## End(Not run)
```

---

nUniqueNeighDons *How many donors are contained among the nearest neighbors?*

Description

This function constructs a variable that for each event shows the number of donors in its nearest neighbor surroundings. It builds on the same idea as has been put forward in the Sconify package: -Burns TJ (2019). Sconify: A toolkit for performing KNN-based statistics for flow and mass cytometry data. R package version 1.4.0 and -Hart GT, Tran TM, Theorell J, Schlums H, Arora G, Rajagopalan S, et al. Adaptive NK cells in people exposed to Plasmodium falciparum correlate with protection from malaria. J Exp Med. 2019 Jun 3;216(6):1280–90. First, the k nearest neighbors are defined for cell x. Then, the number of donors in the k nearest neighbor cloud is returned.

Usage

```
nUniqueNeighDons(
  donorData,
  euclidSpaceData,
  neighRows = seq_len(nrow(as.matrix(donorData))),
  ctrlRows,
  kNeighK = max(100, round(nrow(as.matrix(euclidSpaceData))/10000)),
  kMeansK = max(1, round(nrow(as.matrix(euclidSpaceData))/1000))
)
```

Arguments

donorData	The donor information.
euclidSpaceData	The data cloud in which the nearest neighbors for the events should be identified. Can be a vector, matrix or dataframe.
neighRows	The rows in the dataset that correspond to the neighbors of the donorData points. This can be all the donorData points, or a subset, depending on the setup.
ctrlRows	Optionally, a set of control rows that are used to remove background signal from the neighRows data before sending the data back.
kNeighK	The number of nearest neighbors.
kMeansK	The number of clusters in the initial step of the algorithm. A higher number leads to shorter runtime, but potentially lower accuracy.

Value

An object of the same dimensions as donorData that has been smoothed.

Examples

```
data(testData)
data(testDataSNE)
euclidSpaceData <-
    testData[, c(
        "SYK", "CD16", "CD57", "EAT.2",
        "CD8", "NKG2C", "CD2", "CD56"
    )]
## Not run:
nDonorsVector <- nUniqueNeighDons(
    donorData = as.numeric(testData$label),
    euclidSpaceData
)

## End(Not run)
```

---

testData *A 14 color flow cytometry dataset for example execution and playing around*

Description

This dataset is a 14 color pre-compensated, transformed flow cytometry dataset focusing on cytotoxic lymphocytes, where dead cells have been removed. To make examples very obvious, differences have been artificially exaggregated in the data. The dataset is produced by J. Theorell.

Usage

```
data("testData")
```

Format

An object of class 'data.frame';

`testDataDepeche` *A depeche clustering of the testData set*

Description

This is a depeche clustering of the testData dataset. It has been generated with the Rtsne.multicore package

Usage

```
data(testDataDepeche)
```

Format

An object of class 'list';

Details

produced by J. Theorell.

`testDataSNE` *SNE of the testData set*

Description

This is a t-distributed stochastic neighbor embedding of the testData dataset. It has been generated with the Rtsne.multicore package.

Usage

```
data(testDataSNE)
```

Format

An object of class 'list';

References

Jesse H. Krijthe (2015). Rtsne: T-Distributed Stochastic Neighbor Embedding using a Barnes-Hut Implementation (GitHub)

Index

∗ **#'**
    testDataSNE, 31
∗ **datasets**
    testData, 30
    testDataDepeche, 31
    testDataSNE, 31
∗ **internal**
    microClust, 27

dAllocate, 3
dColorPlot, 4, *7, 8*, *10*, *15*, *19*, *21*, *24*
dColorVector, *5, 6*, 7, *9, 10*, *21*
dContours, 8
dDensityPlot, *6–8*, 9, *15*, *19*, *21*, *24*
depeche, *3*, 11, *21*
DepecheR *(*DepecheR-package*)*, 2
DepecheR-package, 2
dResidualPlot, *6*, *8*, *10*, 14, *19*, *24*
dScale, 16
dSplsda, 17
dViolins, *7*, 20
dWilcox, *6*, *8*, *10*, *15*, 22

groupProbPlot, 25

microClust, 27

neighSmooth, *25*, 27
nUniqueNeighDons, 29

p.adjust, *23*

splsda, *18, 19*

testData, 30
testDataDepeche, 31
testDataSNE, 31