Package 'muma'

July 2, 2014

Type Package

Title Metabolomics Univariate and Multivariate Analysis

Version 1.4

Date 2012-12-2

Author Edoardo Gaude, Francesca Chignola, Dimitrios Spiliotopoulos,Silvia Mari, Andrea Spitaleri and Michela Ghitti

Maintainer Edoardo gaude <egaude541@gmail.com>

Depends car, pdist, pls, gplots, mvtnorm, robustbase, gtools, bitops,caTools, pcaPP, rrcov

Description Preprocessing of high-throughput data (normalization and scalings); Principal Component Analysis with help tool for choosing best-separating principal components and automatic testing for outliers; automatic univariate analysis for parametric and non-parametric data, with generation of specific reports (volcano and box plots); partial least square discriminant analysis (PLS-DA); orthogonal partial least square discriminant analysis (OPLS-DA); Statistical Total Correlation Spectroscopy (STOCSY); Ratio Analysis Nuclear Magnetic Resonance (NMR) Spectroscopy (RANSY).

License GPL-2

Repository CRAN

Date/Publication 2012-12-03 18:20:37

NeedsCompilation no

R topics documented:

muma-package 2
box.plot 3
chose.driver 4

col.pvalues . . . 5
explore.data . . . 7
MetaBc . . . 16
oplsda . . . 19
ostocsy . . . 24
outlier . . . 25
Plot.pca . . . 27
Plot.pca.loading . . . 28
Plot.pca.pvalues . . . 29
Plot.pca.score . . . 31
Plot.plsda . . . 33
plsda . . . 35
pvalues . . . 38
ransy . . . 40
shapiro . . . 42
stocsy . . . 44
stocsy.1d . . . 45
univariate . . . 47
volcano . . . 51
welch . . . 54
wmw . . . 56
work.dir . . . 57

Index **59**

`muma-package` *Metabolomics Univariate and Multivariate Analyses*

Description

Preprocessing of metabolomic data (normalization and scalings); Principal Component Analysis with help tool for choosing best-separating principal components and automatic outlier testing; automatic univariate analysis for parametric and non-parametric data, with report generation (volcano and box plots); partial least sqaure discriminant analysis (PLS-DA); orthogonal partial least square discriminant analysis (OPLS-DA); Statistical Total Correlation Spectroscopy (STOCSY); Ratio Analysis NMR SpectroscopY (RANSY).

Details

Package:	muma
Type:	Package
Version:	1.4
Date:	2012-12-2
License:	GPL-2

Every analysis should start with the function 'explore.data' in order to create the proper files for subsequent analysis. However, it is possible to start the analysis also with the function 'univariate', but, in this case, only univariate analysis can be performed.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Maintainer: Edoardo Gaude <egaude541@gmail.com>

`box.plot` *Generate box plots*

Description

Generates box plots of each variable (column) present in the matrix provided and according to the class specification provided

Usage

```
box.plot(file)
```

Arguments

`file`	a connection or a character string giving the name of the file containing information to plot

Details

This function can be used by calling the function 'univariate', in which 'box.plot' is implemented. The groups are distinguished according to the class vector provided within the dataset. In particular, the file provided has to be formatted with the first column indicating the name of each sample and the second column indicating the class of belonging of each sample.

Value

A list of box plots is generated and written in a directory called 'BoxPlot', within the directory 'Univariate' (generated by the function 'univariate'), in the working directory.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

See Also

univariate

Examples

```
## The function is currently defined as
function (file)
{
    pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
    file = pwdfile
    x <- read.csv(file, sep = ",", header = TRUE)
    x.x = x[, 3:ncol(x)]
    rownames(x.x) = x[, 2]
    k = matrix(x[, 1], ncol = 1)
    colnames(k)[1] = "Class"
    x.x = cbind(k, x.x)
    sorted = x.x[order(x.x[, 1]), ]
    sorted.x = as.matrix(sorted[, -1], ncol = ncol(sorted) -
        1)
    g = c()
    for (i in 1:nrow(sorted)) {
        if (any(g == sorted[i, 1])) {
            g = g
        }
        else {
            g = matrix(c(g, sorted[i, 1]), ncol = 1)
        }
    }
    NoF = nrow(g)
    dirbox = paste(getwd(), "/Univariate/BoxPlot/", sep = "")
    dir.create(dirbox)
    for (i in 2:ncol(x.x)) {
        name = paste(getwd(), "/Univariate/BoxPlot/", colnames(x.x)[i],
            ".pdf", sep = "")
        pdf(name)
        boxplot(x.x[, i] ~ x.x[, 1], boxfill = c(seq(1, NoF)),
            ylab = colnames(x.x)[i], xlab = "Groups", border = "grey30",
            main = paste("Boxplot ", colnames(x.x)[i], sep = ""))
        dev.off()
    }
  }
```

`chose.driver` *List variables in the dataset*

Description

List the variables (column names) of the dataset provided, in order to select the variable against which ransy and/or stocsy1d have to be performed.

Usage

```
chose.driver(scaling)
```

Arguments

scaling	a character indicating the name of the scaling used within the function 'explore.data' (see Details).

Details

The type of scaling to be used has to be indicated within the function 'explore.data' and, when recalling the scaling the same name has to be used. The function 'chose.driver' can be used for listing the names of the variables (column names of the matrix/file provided) and for selecting the variable against which perform stocsy and /or ransy. For details see the functions 'ransy' and 'stocsy.1d'.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (scaling)
{
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling, "/ProcessedTable.csv",
        sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    print(colnames(x.x))
  }
```

---

col.pvalues *Color-coding based on significance*

Description

This function is implemented in the unique function for univariate statistical analysis 'univariate'. This function generates a vector of characters specifying a color-code based on the significance (p-values < 0.05) of each variable.

Usage

```
col.pvalues(file)
```

Arguments

file	a connection or a character string giving the name of the file containing the variables (matrix columns) to test.

Details

For details see the function 'univariate'.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

References

Goodpaster, AM, et al. Statistical Significance analysis of nuclear magnetic resonance-based metabonomics data. (2010) Anal Biochem. 401:134-143.

Examples

```
## The function is currently defined as
function (file)
{
    pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
    file = pwdfile
    x <- read.csv(file, sep = ",", header = TRUE)
    x.x = x[, 3:ncol(x)]
    rownames(x.x) = x[, 2]
    k = matrix(x[, 1], ncol = 1)
    x.n = cbind(k, x.x)
    sorted = x.n[order(x.n[, 1]), ]
    g = c()
    for (i in 1:nrow(sorted)) {
        if (any(g == sorted[i, 1])) {
            g = g
        }
        else {
            g = matrix(c(g, sorted[i, 1]), ncol = 1)
        }
    }
    NoF = nrow(g)
    all.pvalues = matrix(rep(1, ncol(sorted) - 1), ncol = 1)
    dirout.col = paste(getwd(), "/Univariate/Pvalues/", sep = "")
    fin = ncol(sorted) - 1
    for (i in 1:NoF) {
        for (j in 1:NoF) {
            if (i < j) {
                ni = paste("Pvalues_", i, "vs", j, ".csv", sep = "")
                pwdi = paste(getwd(), "/Univariate/Pvalues/",
                  ni, sep = "")
                I = read.csv(pwdi, header = TRUE)
                I = matrix(I[, -1])
                for (q in 1:fin) {
                  if (I[q, ] < 0.05 & all.pvalues[q, ] == 1) {
                    all.pvalues[q, ] = I[q, ]
                  }
```

```
                  else {
                    all.pvalues[q, ] = all.pvalues[q, ]
                  }
                }
            }
        }
    }
    colp = matrix(rep(NA, ncol(sorted) - 1), ncol = 1)
    for (i in 1:fin) {
        if (all.pvalues[i, ] < 1) {
            colp[i, ] = "red"
        }
        else {
            colp[i, ] = "black"
        }
        colnam = "Colors_Pvalues"
        assign(colnam, colp)
        write.csv(colp, paste(dirout.col, colnam, sep = ""))
    }
  }
```

`explore.data` *Preprocessing and principal component analysis of metabolomic data*

Description

Performs normalization and scaling of metabolomics data, providing an overview of the dataset through principal component analysis and automatic outlier testing; it also contains a test for chosing the best-separating principal components

Usage

```
explore.data(file, scaling, scal = TRUE, normalize = TRUE, imputation = FALSE, imput)
```

Arguments

`file`	a connection or a character string giving the name of the file to preprocess and explore
`scaling`	the type of scaling to be used (see Details)
`scal`	logical, whether to perform or not scaling. Default is 'TRUE'. See details.
`normalize`	logical, whether to perform or not normalization. Default is 'TRUE'.
`imputation`	logical, whether to perform or not imputation of missing values. Default is 'FALSE'.
`imput`	character vector indicating the type of value with which missing values should be imputed. See details for the different options.

Details

The 'file' provided has to be a matrix in .csv form, formatted with the first column indicating the name of the samples and the second column indicating the class of belonging of each samples (e.g. treatment groups, healthy/diseased, ...). The header of the matrix must contain the name of each variable in the dataset.

Before performing data preprocessing the function 'explore.data' scans the dataset to find negative values and substitutes them with 0 values. As a result, a table reporting the negative values and a table with corrected values are generated and written in the directory 'Preprocessing_Data'.

There for options for imputing missing values: "mean", "minimum", "half.minimum", "zero". For specifying the type of imput to be used, the field 'imputation' must be turned to 'TRUE'.

A normalization is automatically performed on total intensity, i.e. the sum of all variables for each sample is calculated and used as normalizing factor for each variable. A table reporting normalized valued is generated and written in the directory 'Preprocessing_Data'.

Different types of 'scaling' can be performed. 'pareto', 'Pareto', 'p' or 'P' can be used for specifying the Pareto scaling. 'auto', 'Auto', 'auto', 'a' or 'A' can be used for specifying the Auto scaling (or unit variance scaling). 'vast', 'Vast', 'v' or 'V' can be used for specifying the vast scaling. 'range', 'Range', 'r' or 'R' can be used for specifying the Range scaling. A table reporting the scaled values is generated and written in the directory 'Preprocessing_Data'. If the 'scal' is turned to 'FALSE', no scaling is performed, but only mean-centering.

Principal component analysis is automatically performed on scaled table and a plot reporting pairwise representation of the first ten principal components is graphically visualized and written in the directory 'PCA_Data'. A statistical test is performed for identifying the best-separating couple of principal components and a rank of the first three couples of components is printed. This can allow the user to chose the best set of principal components. Finally, a geometric test is performed for identifying potential outliers; the result of this test is printed.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (file, scaling, scal = TRUE, normalize = TRUE, imputation = FALSE,
    imput)
{
    comp = read.csv(file, sep = ",", header = TRUE)
    comp.x = comp[, 3:ncol(comp)]
    comp.x = cbind(comp[, 2], comp[, 1], comp.x)
    x <- comp.x
    x.x <- x[, 3:ncol(x)]
    rownames(x.x) <- x[, 2]
    if (!scal) {
        scaling = ""
    }
    dirout = paste(getwd(), "/Preprocessing_Data_", scaling,
```

```
    "/", sep = "")
dir.create(dirout)
if (imputation) {
    y = x.x
    r = is.na(y)
    for (k in 1:ncol(r)) {
        vec = matrix(r[, k], ncol = 1)
        who.miss.rows = which(apply(vec, 1, function(i) {
            any(i)
        }))
        if (length(who.miss.rows) > nrow(y) * 0.8) {
            warning(paste("The variable -", colnames(y)[k],
             "- has a number of missing values > 80%, therefore has been eliminated",
              sep = " "))
            y = y[, -k]
        }
    }
    r = is.na(y)
    who.miss.columns = c()
    for (i in 1:nrow(y)) {
        for (j in 1:ncol(y)) {
            if (r[i, j] == TRUE) {
              if (imput == "mean") {
                v2 = matrix(r[, j], ncol = 1)
                who.miss.rows = which(apply(v2, 1, function(i) {
                  any(i)
                }))
                y[i, j] = mean(y[-who.miss.rows, j])
                print(paste("Imputing missing value of variable -",
                  colnames(y)[j], "- for the observation -",
                  rownames(y)[i], "- with", imput, "value",
                  sep = " "))
              }
              else if (imput == "minimum") {
                v2 = matrix(r[, j], ncol = 1)
                who.miss.rows = which(apply(v2, 1, function(i) {
                  any(i)
                }))
                y[i, j] = min(y[-who.miss.rows, j])
                print(paste("Imputing missing value of variable -",
                  colnames(y)[j], "- for the observation -",
                  rownames(y)[i], "- with", imput, "value",
                  sep = " "))
              }
              else if (imput == "half.minimum") {
                v2 = matrix(r[, j], ncol = 1)
                who.miss.rows = which(apply(v2, 1, function(i) {
                  any(i)
                }))
                y[i, j] = min(y[-who.miss.rows, j])/2
                print(paste("Imputing missing value of variable -",
                  colnames(y)[j], "- for the observation -",
                  rownames(y)[i], "- with", imput, "value",
```

```
                    sep = " "))
                }
                else if (imput == "zero") {
                  v2 = matrix(r[, j], ncol = 1)
                  who.miss.rows = which(apply(v2, 1, function(i) {
                    any(i)
                  }))
                  y[i, j] = 0
                  print(paste("Imputing missing value of variable -",
                    colnames(y)[j], "- for the observation -",
                    rownames(y)[i], "- with", imput, "value",
                    sep = " "))
                }
              }
        }
    }
    pwdi = paste(getwd(), "/Preprocessing_Data_", scaling,
        "/ImputedMatrix.csv", sep = "")
    write.csv(y, pwdi)
    x.x = y
}
for (i in 1:nrow(x.x)) {
    for (j in 1:ncol(x.x)) {
        if (x.x[i, j] <= 0) {
            x.x[i, j] = runif(1, 0, 1e-10)
        }
    }
}
x.x = cbind(comp[, 2], x.x)
write.csv(x.x, paste(dirout, "CorrectedTable.csv", sep = ""))
pwd.c = paste(getwd(), "/Preprocessing_Data_", scaling, "/CorrectedTable.csv",
    sep = "")
x <- read.csv(pwd.c, sep = ",", header = TRUE)
x.x <- x[, 3:ncol(x)]
rownames(x.x) <- x[, 1]
k = matrix(x[, 2], ncol = 1)
if (normalize) {
    x.t <- t(x.x)
    x.s <- matrix(colSums(x.t), nrow = 1)
    uni = matrix(rep(1, nrow(x.t)), ncol = 1)
    area.uni <- uni %*% x.s
    x.areanorm <- x.t/area.uni
    x.areanorm = t(x.areanorm)
    write.csv(x.areanorm, paste(dirout, "/ProcessedTable.csv",
        sep = ""))
}
else {
    write.csv(x.x, paste(dirout, "/ProcessedTable.csv", sep = ""))
}
if (scal) {
    if (scaling == "Pareto" | scaling == "pareto" | scaling ==
        "P" | scaling == "p") {
        pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling,
```

```
        "/ProcessedTable.csv", sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.areanorm.tc <- scale(x.x, center = TRUE, scale = FALSE)
    all.sd <- matrix(apply(x.areanorm.tc, 2, sd), nrow = 1)
    uni.exp.all = matrix(rep(1, nrow(x.areanorm.tc)),
        ncol = 1)
    all.sdm = uni.exp.all %*% all.sd
    all.sqsd = sqrt(all.sdm)
    all.pareto <- x.areanorm.tc/all.sqsd
    write.csv(all.pareto, paste(dirout, "/ProcessedTable.csv",
        sep = ""))
}
else if (scaling == "Auto" | scaling == "auto" | scaling ==
    "A" | scaling == "a") {
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling,
        "/ProcessedTable.csv", sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.areanorm.tc <- scale(x.x, center = TRUE, scale = FALSE)
    all.sd <- matrix(apply(x.areanorm.tc, 2, sd), nrow = 1)
    uni.exp.all = matrix(rep(1, nrow(x.areanorm.tc)),
        ncol = 1)
    all.sdm = uni.exp.all %*% all.sd
    all.auto <- x.areanorm.tc/all.sdm
    write.csv(all.auto, paste(dirout, "/ProcessedTable.csv",
        sep = ""))
}
else if (scaling == "Vast" | scaling == "vast" | scaling ==
    "V" | scaling == "v") {
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling,
        "/ProcessedTable.csv", sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.areanorm.tc <- scale(x.x, center = TRUE, scale = FALSE)
    all.sd <- matrix(apply(x.areanorm.tc, 2, sd), nrow = 1)
    uni.exp.all = matrix(rep(1, nrow(x.areanorm.tc)),
        ncol = 1)
    all.sdm = uni.exp.all %*% all.sd
    sdm2 = all.sdm^2
    colm = matrix(colMeans(x.x), nrow = 1)
    colm.m = uni.exp.all %*% colm
    num = x.areanorm.tc * colm.m
    vast = num/sdm2
    write.csv(vast, paste(dirout, "/ProcessedTable.csv",
        sep = ""))
}
else if (scaling == "Range" | scaling == "range" | scaling ==
    "R" | scaling == "r") {
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling,
```

```
            "/ProcessedTable.csv", sep = "")
        x <- read.csv(pwd.n, sep = ",", header = TRUE)
        x.x <- x[, 2:ncol(x)]
        rownames(x.x) <- x[, 1]
        x.areanorm.tc <- scale(x.x, center = TRUE, scale = FALSE)
        range = c()
        for (i in 1:ncol(x.x)) {
            den = c()
            den = max(x.x[, i]) - min(x.x[, i])
            range = matrix(c(range, den), nrow = 1)
        }
        uni.exp.all = matrix(rep(1, nrow(x.areanorm.tc)),
            ncol = 1)
        range.m = uni.exp.all %*% range
        all.range = x.areanorm.tc/range.m
        write.csv(all.range, paste(dirout, "/ProcessedTable.csv",
            sep = ""))
    }
    else if (scaling == "Median" | scaling == "median" |
        scaling == "M" | scaling == "m") {
        pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling,
            "/ProcessedTable.csv", sep = "")
        x <- read.csv(pwd.n, sep = ",", header = TRUE)
        x.x <- x[, 2:ncol(x)]
        rownames(x.x) <- x[, 1]
        x.areanorm.tc <- scale(x.x, center = TRUE, scale = FALSE)
        all.med <- matrix(apply(x.areanorm.tc, 2, median),
            nrow = 1)
        uni.exp.all = matrix(rep(1, nrow(x.areanorm.tc)),
            ncol = 1)
        all.sdm = uni.exp.all %*% all.med
        all.med <- x.areanorm.tc/all.sdm
        write.csv(all.med, paste(dirout, "/ProcessedTable.csv",
            sep = ""))
    }
}
else {
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling,
        "/ProcessedTable.csv", sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.c = scale(x.x, scale = FALSE)
    write.csv(x.c, paste(dirout, "/ProcessedTable.csv", sep = ""))
}
pwd.scal = paste(getwd(), "/Preprocessing_Data_", scaling,
    "/ProcessedTable.csv", sep = "")
x <- read.csv(pwd.scal, sep = ",", header = TRUE)
x.x <- x[, 2:ncol(x)]
rownames(x.x) <- x[, 1]
pc.all <- prcomp(x.x, center = FALSE, scale = FALSE)
p.v <- matrix(((pc.all$sdev^2)/(sum(pc.all$sdev^2))), ncol = 1)
p.i <- round(p.v * 100, 1)
```

```
p.z <- matrix(1, nrow(p.i), 1)
p.f <- cbind(p.i, p.z)
dirout.pca = paste(getwd(), "/PCA_Data_", scaling, "/", sep = "")
dir.create(dirout.pca)
write.csv(p.f, paste(dirout.pca, "PCA_P", sep = ""))
write.csv(pc.all$x, paste(dirout.pca, "PCA_ScoreMatrix.csv",
    sep = ""))
write.csv(pc.all$rotation, paste(dirout.pca, "PCA_LoadingsMatrix.csv",
    sep = ""))
pwd.score = paste(getwd(), "/PCA_Data_", scaling, "/", "PCA_ScoreMatrix.csv",
    sep = "")
Score <- read.csv(pwd.score, sep = ",", header = TRUE)
Score.x <- Score[, 2:ncol(Score)]
rownames(Score.x) <- Score[, 1]
pwd.load = paste(getwd(), "/PCA_Data_", scaling, "/", "PCA_LoadingsMatrix.csv",
    sep = "")
Loading <- read.csv(pwd.load, sep = ",", header = TRUE)
Loading.x <- Loading[, 2:ncol(Loading)]
rownames(Loading.x) <- Loading[, 1]
pwd.pvar = paste(getwd(), "/PCA_Data_", scaling, "/", "PCA_P",
    sep = "")
Pvar <- read.csv(pwd.pvar, sep = ",", header = TRUE)
Pvar.x <- Pvar[, 2:ncol(Pvar)]
rownames(Pvar.x) <- Pvar[, 1]
barplot(Pvar.x[, 1], xlab = "Principal Components", ylab = "Proportion of Variance explained",
    main = "Screeplot", ylim = c(0, 100))
scree = paste(dirout.pca, "Screeplot", scaling, ".pdf", sep = "")
dev.copy2pdf(file = scree)
tutticolors = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, "rosybrown4",
    "green4", "navy", "purple2", "orange", "pink", "chocolate2",
    "coral3", "khaki3", "thistle", "turquoise3", "palegreen1",
    "moccasin", "olivedrab3", "azure4", "gold3", "deeppink"),
    ncol = 1)
col = c()
for (i in 1:nrow(k)) {
    col = c(col, tutticolors[k[i, ], ])
}
pairs = c()
if (ncol(Score.x) >= 10) {
    pairs = c(10)
}
else {
    pairs = c(ncol(Score.x))
}
pairs(Score.x[, 1:pairs], col = col)
dev.new()
pairs = paste(dirout.pca, "First_10_Components_", scaling,
    ".pdf", sep = "")
dev.copy2pdf(file = pairs)
K = paste(getwd(), "/Preprocessing_Data_", scaling, "/class.csv",
    sep = "")
write.csv(k, K)
x.nn = cbind(k, pc.all$x)
```

```
sorted = x.nn[order(x.nn[, 1]), ]
g = c()
for (i in 1:nrow(sorted)) {
    if (any(g == sorted[i, 1])) {
        g = g
    }
    else {
        g = matrix(c(g, sorted[i, 1]), ncol = 1)
    }
}
dirout.g = paste(getwd(), "/Groups", sep = "")
dir.create(dirout.g)
for (i in 1:nrow(g)) {
    vuota <- c()
    fin = matrix(rep(NA, ncol(sorted)), nrow = 1)
    for (j in 1:nrow(sorted)) {
        if (sorted[j, 1] == i) {
            vuota <- matrix(sorted[j, ], nrow = 1)
            rownames(vuota) = rownames(sorted)[j]
            fin = rbind(fin, vuota)
        }
    }
    nam = paste("r", i, sep = ".")
    n = matrix(fin[-1, ], ncol = ncol(sorted))
    n.x = matrix(n[, -1], ncol = ncol(sorted) - 1)
    name = as.matrix(assign(nam, n.x))
    outputfileg = paste("r.", i, ".csv", sep = "")
    write.csv(name, paste(dirout.g, outputfileg, sep = "/"),
        row.names = FALSE)
}
all.F = c()
NoF = nrow(g)
for (i in 1:NoF) {
    for (j in 1:NoF) {
        if (i < j) {
            ni = paste("r.", i, ".csv", sep = "")
            nj = paste("r.", j, ".csv", sep = "")
            pwdi = paste(getwd(), "/Groups/", ni, sep = "")
            pwdj = paste(getwd(), "/Groups/", nj, sep = "")
            I = read.csv(pwdi, header = TRUE)
            J = read.csv(pwdj, header = TRUE)
            fin = ncol(I) - 1
            library(rrcov)
            ntest = factorial(fin)/(2 * (factorial(fin -
              2)))
            T2 = c()
            nam = c()
            for (k in 1:fin) {
              for (l in 1:fin) {
                if (k < l) {
                  Ikl = cbind(I[, k], I[, l])
                  Jkl = cbind(J[, k], J[, l])
                  t1 = matrix(T2.test(Ikl, Jkl)$statistic,
```

```
                    ncol = 2)
                t2 = c(t1[, 1])
                T2 = matrix(c(T2, t2), ncol = 1)
                rownam = paste("PC", k, "vsPC", l, sep = "")
                nam = matrix(c(nam, rownam), ncol = 1)
              }
            }
          }
          pair = paste("T2statistic_", i, "vs", j, sep = "")
          rownames(T2) = nam
          colnames(T2)[1] = pair
          num = nrow(I) + nrow(J) - 3
          den = 2 * (nrow(I) + nrow(J) - 2)
          coeff = num/den
          Fval = T2 * coeff
          Fvalname = paste("F_statistic_", i, "vs", j,
            sep = "")
          colnames(Fval)[1] = Fvalname
          Fpval = pf(Fval, 2, num)
          Fname = paste("F_pvalue_", i, "vs", j, sep = "")
          colnames(Fpval)[1] = Fname
          Fpvalfin = 1 - Fpval
          all.F = matrix(c(all.F, Fpvalfin))
        }
    }
}
varp = c()
for (k in 1:fin) {
    for (l in 1:fin) {
        if (k < l) {
            varp = matrix(c(varp, p.f[k, 1] + p.f[l, 1]),
              ncol = 1)
        }
    }
}
ncomparison = factorial(nrow(g))/(2 * (factorial(nrow(g) -
    2)))
all.F = matrix(all.F, ncol = ncomparison)
rownames(all.F) = nam
allFpwd = paste(getwd(), "/PCA_Data_", scaling, "/PCs_Fstatistic.csv",
    sep = "")
write.csv(all.F, allFpwd, row.names = FALSE)
sum = matrix(rowSums(all.F), ncol = 1)
all = data.frame(nam, sum, varp)
colnames(all)[3] = "Variance(%)"
colnames(all)[2] = "Sum_p_values"
colnames(all)[1] = "Pair_of_PCs"
ord.sum = all[order(all[, 2]), ]
colnames(ord.sum)[3] = "Variance(%)"
colnames(ord.sum)[2] = "Sum_p_values(F_statistics)"
colnames(ord.sum)[1] = "Pair_of_PCs"
rownames(ord.sum) = 1:nrow(ord.sum)
rankFpwd = paste(getwd(), "/PCA_Data_", scaling, "/PCs_ranked_Fpvalue.csv",
```

```
        sep = "")
    write.csv(ord.sum, rankFpwd, row.names = FALSE)
   print("Pairs of Principal Components giving highest statistical cluster separation are:")
    print(ord.sum[1:5, ])
  }
```

MetaBc *Metabolomics data of B cells*

Description

Data from metabolomics analysis of B cells harvested in different conditions and at different time points

Usage

```
data(MetaBc)
```

Format

A data frame with 25 observations on the following 100 variables.

`Samples` a factor with levels `-LPS1` `-LPS2` `-LPS3` `-LPS4` `-LPS5` `DAY1.1` `DAY1.2` `DAY1.3` `DAY1.4` `DAY1.5` `DAY2.1` `DAY2.2` `DAY2.3` `DAY2.4` `DAY2.5` `DAY3.1` `DAY3.2` `DAY3.3` `DAY3.4` `DAY3.5` `DAY4.1` `DAY4.2` `DAY4.3` `DAY4.4` `DAY4.5`

`Class` a numeric vector

`X0.89` a numeric vector

`X0.93` a numeric vector

`X0.95` a numeric vector

`X0.98` a numeric vector

`X1` a numeric vector

`X1.03` a numeric vector

`X1.11` a numeric vector

`X1.17` a numeric vector

`X1.17.1` a numeric vector

`X1.21` a numeric vector

`X1.24` a numeric vector

`X1.3` a numeric vector

`X1.32` a numeric vector

`X1.43` a numeric vector

`X1.47` a numeric vector

`X1.56` a numeric vector

`X1.58` a numeric vector

`X1.64` a numeric vector

`X1.71` a numeric vector

`X1.71.1` a numeric vector

`X1.89` a numeric vector

`X1.9` a numeric vector

`X2.03` a numeric vector

`X2` a numeric vector

`X2.06` a numeric vector

`X2.11` a numeric vector

`X2.22` a numeric vector

`X2.34` a numeric vector

`X2.37` a numeric vector

`X2.39` a numeric vector

`X2.44` a numeric vector

`X2.53` a numeric vector

`X2.51` a numeric vector

`X2.49` a numeric vector

`X2.65` a numeric vector

`X2.73` a numeric vector

`X2.8` a numeric vector

`X2.84` a numeric vector

`X2.91` a numeric vector

`X3.03` a numeric vector

`X3.13` a numeric vector

`X3.19` a numeric vector

`X3.24` a numeric vector

`X3.23` a numeric vector

`X3.27` a numeric vector

`X3.26` a numeric vector

`X3.33` a numeric vector

`X3.38` a numeric vector

`X3.39` a numeric vector

`X3.4` a numeric vector

`X3.41` a numeric vector

`X3.41.1` a numeric vector

`X3.42` a numeric vector

X3.45 a numeric vector

X3.48 a numeric vector

X3.47 a numeric vector

X3.49 a numeric vector

X3.5 a numeric vector

X3.53 a numeric vector

X3.56 a numeric vector

X3.61 a numeric vector

X3.65 a numeric vector

X3.63 a numeric vector

X3.67 a numeric vector

X3.69 a numeric vector

X3.69.1 a numeric vector

X3.72 a numeric vector

X3.77 a numeric vector

X3.83 a numeric vector

X3.81 a numeric vector

X3.8 a numeric vector

X3.85 a numeric vector

X3.84 a numeric vector

X3.91 a numeric vector

X3.89 a numeric vector

X3.95 a numeric vector

X4 a numeric vector

X4.03 a numeric vector

X4.06 a numeric vector

X4.11 a numeric vector

X4.17 a numeric vector

X4.24 a numeric vector

X4.33 a numeric vector

X5.38 a numeric vector

X6.51 a numeric vector

X6.88 a numeric vector

X7.04 a numeric vector

X7.18 a numeric vector

X7.32 a numeric vector

X7.41 a numeric vector

`X7.36` a numeric vector

`X7.53` a numeric vector

`X7.74` a numeric vector

`X8.1` a numeric vector

`X8.19` a numeric vector

`X8.44` a numeric vector

`X8.71` a numeric vector

`X8.93` a numeric vector

Details

Variables are named as NMR chemical shifts and values are the areas of NMR peaks, at any given chemical shift. The first column of the dataset represents the name of each sample and is informative of sample condition/treatment. The second column of the dataset reports the information regarding the belonging of each sample to a specific class of condition/treatment.

Source

Garcia-Manteiga, J.M. Metabolomics of B to Plasma Cell Differentiation. (2011) J Proteom Res.10(9)4165-4176.

`oplsda` *Orthogonal Partial Least Sqaure Discriminant Analysis*

Description

Partial Least Square Discriminant Analysis performed with the OSC filter

Usage

```
oplsda(scaling)
```

Arguments

`scaling`	a character string indicating the name of the scaling previously used with the function 'explore.data'

Details

The element 'scaling' is needed to identify which scaled table must be used in the opls analysis. OPLSDA score plot and corresponding S-plot are graphically visualized and written in the directory 'OPLSDA'.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

References

Bylesjo, M. et al. OPLS discriminant analysis: combining the strenghts of PLS-DA and SIMCA classification. (2006) J Chemometrics. 20:341-351.

Examples

```
## The function is currently defined as
function (scaling)
{
    pwd.x = paste(getwd(), "/Preprocessing_Data_", scaling, "/ProcessedTable.csv",
        sep = "")
    x = read.csv(pwd.x, header = TRUE)
    x.x = x[, 2:ncol(x)]
    rownames(x.x) = x[, 1]
    pwdK = paste(getwd(), "/Preprocessing_Data_", scaling, "/class.csv",
        sep = "")
    k = read.csv(pwdK, header = TRUE)
    k.x = matrix(k[, -1], ncol = 1)
    x.n = cbind(k.x, x.x)
    sorted = x.n[order(x.n[, 1]), ]
    k = matrix(sorted[, 1], ncol = 1)
    g = c()
    for (i in 1:nrow(sorted)) {
        if (any(g == sorted[i, 1])) {
            g = g
        }
        else {
            g = matrix(c(g, sorted[i, 1]), ncol = 1)
        }
    }
    Y = matrix(rep(NA, nrow(sorted)), ncol = 1)
    for (i in 1:nrow(sorted)) {
        for (l in 1:2) {
            if (sorted[i, 1] == l) {
                Y[i, ] = 0
            }
            else {
                Y[i, ] = 1
            }
        }
    }
    X = as.matrix(sorted[, -1], ncol = ncol(sorted) - 1)
    nf = 1
    T = c()
    P = c()
    C = c()
    W = c()
    Tortho = c()
    Portho = c()
    Wortho = c()
    Cortho = c()
```

```
for (j in 1:nf) {
    w = (t(X) %*% Y) %*% solve(t(Y) %*% Y)
    w1 = t(w) %*% w
    w2 = abs(sqrt(w1))
    w = w %*% solve(w2)
    t = (X %*% w) %*% solve(t(w) %*% w)
    t1 = t(t) %*% t
    c = t(Y) %*% t %*% solve(t1)
    c1 = t(c) %*% c
    u = Y %*% c %*% solve(c1)
    u1 = t(u) %*% u
    u2 = abs(sqrt(u1))
    p = (t(X) %*% t) %*% solve(t1)
    wortho = p - w
    wortho1 = t(wortho) %*% wortho
    wortho2 = abs(sqrt(abs(wortho1)))
    wortho = wortho %*% solve(wortho2)
    tortho = X %*% wortho %*% solve(t(wortho) %*% wortho)
    tortho1 = t(tortho) %*% tortho
    portho = t(X) %*% tortho %*% solve(tortho1)
    cortho = t(Y) %*% tortho %*% solve(tortho1)
    X = X - tortho %*% t(portho)
    T = matrix(c(T, t))
    P = matrix(c(P, p))
    C = matrix(c(C, c))
    W = matrix(c(W, w))
    Tortho = matrix(c(Tortho, tortho))
    Portho = matrix(c(Portho, portho))
    Wortho = matrix(c(Wortho, wortho))
    Cortho = matrix(c(Cortho, cortho))
}
T = matrix(T, ncol = nf)
T = scale(T, scale = FALSE, center = TRUE)
P = matrix(P, ncol = nf)
C = matrix(C, ncol = nf)
W = matrix(W, ncol = nf)
Tortho = matrix(Tortho, ncol = nf)
Portho = matrix(Portho, ncol = nf)
Cortho = matrix(Cortho, ncol = nf)
Wortho = matrix(Wortho, ncol = nf)
Xortho = Tortho %*% t(Portho)
max.pc1 = 1.3 * (max(abs(T[, nf])))
max.pc2 = 1.3 * (max(abs(Tortho[, nf])))
lim = c()
if (max.pc1 > max.pc2) {
    lim = c(-max.pc1, max.pc1)
}
else {
    lim = c(-max.pc2, max.pc2)
}
tutticolors = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, "rosybrown4",
    "green4", "navy", "purple2", "orange", "pink", "chocolate2",
    "coral3", "khaki3", "thistle", "turquoise3", "palegreen1",
```

```
    "moccasin", "olivedrab3", "azure4", "gold3", "deeppink"),
    ncol = 1)
col = c()
for (i in 1:nrow(k)) {
    col = c(col, tutticolors[k[i, ], ])
}
plot(T[, nf], Tortho[, 1], col = col, pch = 19, xlim = lim,
    ylim = lim, xlab = "T score [1]", ylab = "Orthogonal T score [1]",
    main = "OPLS-DA score scatter plot")
text(T[, nf], Tortho[, 1], col = col, labels = rownames(sorted),
    cex = 0.5, pos = 1)
axis(1, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
axis(2, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
library(car)
dataEllipse(T[, nf], Tortho[, 1], levels = c(0.95), add = TRUE,
    col = "black", lwd = 0.4, plot.points = FALSE, center.cex = 0.2)
dirout = paste(getwd(), "/OPLS-DA", scaling, "/", sep = "")
dir.create(dirout)
pwdxdef = paste(dirout, "X_deflated.csv", sep = "")
write.csv(X, pwdxdef)
scor = paste(dirout, "ScorePlot_OPLS-DA_", scaling, ".pdf",
    sep = "")
dev.copy2pdf(file = scor)
pwdT = paste(dirout, "TScore_Matrix.csv", sep = "")
write.csv(T, pwdT)
pwdTortho = paste(dirout, "TorthoScore_Matrix.csv", sep = "")
write.csv(T, pwdTortho)
s = as.matrix(sorted[, -1], ncol = ncol(sorted) - 1)
p1 = c()
for (i in 1:ncol(s)) {
    scov = cov(s[, i], T)
    p1 = matrix(c(p1, scov), ncol = 1)
}
pcorr1 = c()
for (i in 1:nrow(p1)) {
    den = apply(T, 2, sd) * sd(s[, i])
    corr1 = p1[i, ]/den
    pcorr1 = matrix(c(pcorr1, corr1), ncol = 1)
}
pwdp1 = paste(dirout, "p1_Matrix.csv", sep = "")
write.csv(p1, pwdp1)
pwdpcorr1 = paste(dirout, "pcorr1_Matrix.csv", sep = "")
write.csv(pcorr1, pwdpcorr1)
dev.new()
plot(p1, pcorr1, xlab = "p[1]", ylab = "p(corr)[1]", main = paste("S-plot (OPLS-DA) ",
    scaling, sep = ""))
text(p1, pcorr1, labels = colnames(s), cex = 0.5, pos = 1)
splot = paste(dirout, "SPlot_OPLS-DA_", scaling, ".pdf",
    sep = "")
dev.copy2pdf(file = splot)
pc.all <- prcomp(X, center = FALSE, scale = FALSE)
```

```
p.v <- matrix(((pc.all$sdev^2)/(sum(pc.all$sdev^2))), ncol = 1)
p.i <- round(p.v * 100, 1)
p.z <- matrix(1, nrow(p.i), 1)
p.f <- cbind(p.i, p.z)
dirout.pca = paste(dirout, "PCA_OPLS/", sep = "")
dir.create(dirout.pca)
write.csv(p.f, paste(dirout.pca, "PCA_P_OPLS", sep = ""))
write.csv(pc.all$x, paste(dirout.pca, "PCA_OPLS_ScoreMatrix.csv",
    sep = ""))
write.csv(pc.all$rotation, paste(dirout.pca, "PCA_OPLS_LoadingsMatrix.csv",
    sep = ""))
cum = p.f[1, 1] + p.f[2, 1]
lim = c()
max.pc1 = 1.3 * (max(abs(pc.all$x[, 1])))
max.pc2 = 1.3 * (max(abs(pc.all$x[, 2])))
if (max.pc1 > max.pc2) {
    lim = c(-max.pc1, max.pc1)
}
else {
    lim = c(-max.pc2, max.pc2)
}
pca <- paste("PC1 (", p.f[1, 1], ") %")
pcb <- paste("PC2 (", p.f[2, 1], ") %")
xlab = c(pca)
ylab = c(pcb)
D <- paste(dirout.pca, "PCA_OPLS_ScorePlot.pdf", sep = "")
pdf(D)
plot(pc.all$x[, 1], pc.all$x[, 2], col = col, xlab = xlab,
  ylab = ylab, xlim = lim, ylim = lim, pch = 19, sub = paste("Cumulative Proportion of Variance Explained = ",
       cum, "%", sep = ""), main = "PCA Score Plot on orthogonal-deflated X")
axis(1, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
axis(2, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
library(car)
dataEllipse(pc.all$x[, 1], pc.all$x[, 2], levels = c(0.95),
    add = TRUE, col = "black", lwd = 0.4, plot.points = FALSE,
    center.cex = 0.2)
text(pc.all$x[, 1], pc.all$x[, 2], col = col, cex = 0.5,
    labels = rownames(sorted), pos = 1)
dev.off()
pca.load <- paste("Loading PC1 (", p.f[1, 1], ") %")
pcb.load <- paste("Loading PC2 (", p.f[2, 1], ") %")
Max.pc1 = 1.1 * (max(pc.all$rotation[, 1]))
Min.pc1 = 1.1 * (min(pc.all$rotation[, 1]))
Mpc1 = c(Min.pc1 * 2, Max.pc1 * 2)
Max.pc2 = 1.1 * (max(pc.all$rotation[, 2]))
Min.pc2 = 1.1 * (min(pc.all$rotation[, 2]))
Mpc2 = c(Min.pc2 * 2, Max.pc2 * 2)
E = paste(dirout.pca, "PCA_OPLS_LoadingPlot.pdf", sep = "")
pdf(E)
plot(pc.all$rotation[, 1], pc.all$rotation[, 2], xlab = pca.load,
    ylab = pcb.load, xlim = c(Min.pc1, Max.pc1), ylim = c(Min.pc2,
```

```
        Max.pc2), main = "PCA Loading Plot on orthogonal-deflated X",
      sub = paste("Cumulative Proportion of Variance Explained = ",
        cum, "%", sep = ""))
    axis(1, at = Mpc1, pos = c(0, 0), labels = FALSE, col = "grey",
      lwd = 0.7)
    axis(2, at = Mpc2, pos = c(0, 0), labels = FALSE, col = "grey",
      lwd = 0.7)
    text(pc.all$rotation[, 1], pc.all$rotation[, 2], labels = rownames(pc.all$rotation),
      cex = 0.6, pos = 1)
    dev.off()
  }
```

`ostocsy` *OSC Statistical TOtal Correlation SpectroscopY*

Description

Performs the stocsy analysis on the matrix deflated through the 'oplsda' function

Usage

```
ostocsy(threshold = TRUE, pos.threshold, neg.threshold)
```

Arguments

`threshold`	logical, indicating whether positive and negative threshold must be specified. By default is 'TRUE'.
`pos.threshold`	The positive threshold for plotting positive correlations.
`neg.threshold`	The negative threshold for plotting negative correlations.

Details

OSTOCSY correlation matrix is graphically visualized, with correlations ranging from -1 to 1 and color coded for negative correlations (blue), positive correlations (red) and no correlation (white). If 'threshold' is 'TRUE' an additional plot is generated with correlation values >= and =< to the specified positive and negative thresholds, respectively. All plots visualized are written in the directory 'OSTOCSY', together with the deflated (OSC-filtered) matrix.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

References

Blaise, B.J. et al. Orthogonal filtered recoupled-STOCSY to extract metabolic networkd associated with minor perturbations from NMR spectroscopy. (2011) J Proteome Res. 10(9):4342-8.

Examples

```
## The function is currently defined as
function (threshold = TRUE, pos.threshold, neg.threshold)
{
    pwd.n = paste(getwd(), "/OPLS-DA/X_deflated.csv", sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.t <- x.x
    mycor = cor(x.t, method = c("pearson"))
    library(gplots)
    col = colorpanel(50, "blue", "white", "red")
    image(mycor, axes = FALSE, col = col, main = "OSTOCSY")
   axis(side = 1, labels = colnames(mycor), at = seq(0, 1, length = length(colnames(mycor))),
        las = 2, cex.axis = 0.4)
   axis(side = 2, labels = colnames(mycor), at = seq(0, 1, length = length(colnames(mycor))),
        las = 2, cex.axis = 0.4)
    dirout = paste(getwd(), "/OPLS-DA/OSTOCSY/", sep = "")
    dir.create(dirout)
    o = paste(dirout, "OSTOCSY.pdf", sep = "")
    dev.copy2pdf(file = o)
    o.cor = paste(dirout, "CorrelationMatrix.csv", sep = "")
    write.csv(mycor, file = o.cor)
    if (threshold) {
        dev.new()
        image(mycor, axes = FALSE, zlim = c(pos.threshold, 1),
            col = "red", main = paste("OSTOCSY <", neg.threshold,
                " & >", pos.threshold, sep = ""))
        image(mycor, axes = FALSE, zlim = c(-1, neg.threshold),
            col = "navy", add = TRUE)
        axis(side = 1, labels = colnames(mycor), at = seq(0,
            1, length = length(colnames(mycor))), las = 2, cex.axis = 0.4)
        axis(side = 2, labels = colnames(mycor), at = seq(0,
            1, length = length(colnames(mycor))), las = 2, cex.axis = 0.4)
        out = paste(dirout, "OSTOCSY_", pos.threshold, "_", neg.threshold,
            ".pdf", sep = "")
        dev.copy2pdf(file = out)
    }
  }
```

`outlier` *Test for Outlier*

Description

This function is implemented in the unique function 'plot.pca'. It performs an test for outlier based on geometric distances of each point in the PCA score plot from a 95

Usage

```
outlier(pcx, pcy, scaling)
```

Arguments

pcx	an integer indicating the principal component to be plotted in x.
pcy	an integer indicating the principal component to be plotted in y
scaling	a character string indicating the name of the scaling previously specified in the function 'explore.data'

Details

For details see ?plot.pca.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (pcx, pcy, scaling)
{
    pwd.score = paste(getwd(), "/PCA_Data_", scaling, "/PCA_ScoreMatrix.csv",
        sep = "")
    Score <- read.csv(pwd.score, sep = ",", header = TRUE)
    Score.x <- Score[, 2:ncol(Score)]
    rownames(Score.x) <- Score[, 1]
    dx = scale(Score.x[, pcx], scale = FALSE)
    dy = scale(Score.x[, pcy], scale = FALSE)
    sumdxdx = sum(dx * dx)
    sumdydy = sum(dy * dy)
    sumdxdy = sum(dx * dy)
    theta = 0.5 * atan((2 * sumdxdy)/(sumdydy - sumdxdx))
    c = cos(theta)
    s = sin(theta)
    X = (c * dx) - (s * dy)
    Y = (s * dx) + (c * dy)
    varX = var(X)
    varY = var(Y)
    M = sqrt(varX)
    m = sqrt(varY)
    M95 = M * 3.03315
    m95 = m * 3.03315
    Fx = sqrt(abs((M95^2) - (m95^2)))
    Fy = Fx * tan(theta)
    F1 = c(-Fx, -Fy)
    F2 = c(Fx, Fy)
```

```
    one = matrix(rep(1, nrow(Score.x)), ncol = 1)
    F1.m = one %*% F1
    F2.m = one %*% F2
    library(pdist)
    Punti = cbind(dx, dy)
    dist1 = pdist(Punti[, 1:2], F1.m)
    dist2 = pdist(Punti[, 1:2], F2.m)
    D = matrix(dist1[, 1] + dist2[, 1], ncol = 1)
    v = M95 * 2
    outliers = c()
    O = paste(getwd(), "/PCA_Data_", scaling, "/Outliers_PC",
        pcx, "vs", pcy, ".csv", sep = "")
    write.csv(outliers, O)
    cat("The following observations are calculated as outliers \n",
        file = O)
    for (i in 1:nrow(D)) {
        if (D[i, ] > v) {
            cat(rownames(Score.x)[i], " \n", file = O, append = TRUE)
        }
    }
    outlierfile = read.csv(O, header = TRUE)
    n = nrow(outlierfile)
    if (n == 0) {
        print("No outliers are detected")
    }
    else {
        print(outlierfile)
    }
}
```

`Plot.pca` *principal component analysis plotting*

Description

Visualize PCA score and loading plots.

Usage

```
Plot.pca(pcx, pcy, scaling, test.outlier = TRUE)
```

Arguments

`pcx`	an integer indicating the principal component to be plotted in x
`pcy`	an integer indicating the principal component to be plotted in y
`scaling`	a character string indicating the name of the scaling previously specified in the function 'explore.data'
`test.outlier`	logical, indicating whether the geometric outlier testing has to be performed. By default is 'TRUE'.

Details

'test.outlier' results in a printed string indicating whether outliers were detected or not; if detected, the samples (rownames) identified as outliers are printed. Principal components to be plotted can be identified through the function 'explore.data'.

A directory called 'PCA-Data' is automatically created in the working directory. Within this directory are written PCA score and loading matrix and every PCA plot generated with the function 'plot.pca'.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (pcx, pcy, scaling, test.outlier = TRUE)
{
    Plot.pca.score(pcx, pcy, scaling)
    Plot.pca.loading(pcx, pcy, scaling)
    if (test.outlier) {
        outlier(pcx, pcy, scaling)
    }
  }
```

Plot.pca.loading *Plot PCA Loading Plot*

Description

This function is implemented in the unique function 'plot.pca'. It generated and visualizes PCA loading plot. This function cannot be used without having previously used the function 'explore.data'.

Usage

```
Plot.pca.loading(pcx, pcy, scaling)
```

Arguments

pcx	an integer indicating the principal component to be plotted in x.
pcy	an integer indicating the principal component to be plotted in y
scaling	a character string indicating the name of the scaling previously specified in the function 'explore.data'

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (pcx, pcy, scaling)
{
    loadi = paste(getwd(), "/PCA_Data_", scaling, "/PCA_LoadingsMatrix.csv",
        sep = "")
    Loading <- read.csv(loadi, sep = ",", header = TRUE)
    Loading.x <- Loading[, 2:ncol(Loading)]
    rownames(Loading.x) <- Loading[, 1]
    ppppp = paste(getwd(), "/PCA_Data_", scaling, "/PCA_P", sep = "")
    Pvar <- read.csv(ppppp, sep = ",", header = TRUE)
    Pvar.x <- Pvar[, 2:ncol(Pvar)]
    rownames(Pvar.x) <- Pvar[, 1]
    cum = Pvar[pcx, 2] + Pvar[pcy, 2]
    pca <- paste("Loadings PC", pcx, " (", Pvar[pcx, 2], ") %")
    pcb <- paste("Loadings PC", pcy, " (", Pvar[pcy, 2], ")%")
    lim.load = c()
    Max.pc1 = 1.1 * (max(Loading.x[, pcx]))
    Min.pc1 = 1.1 * (min(Loading.x[, pcx]))
    Mpc1 = c(Min.pc1 * 2, Max.pc1 * 2)
    Max.pc2 = 1.1 * (max(Loading.x[, pcy]))
    Min.pc2 = 1.1 * (min(Loading.x[, pcy]))
    Mpc2 = c(Min.pc2 * 2, Max.pc2 * 2)
    dev.new()
    plot(Loading.x[, pcx], Loading.x[, pcy], xlab = pca, ylab = pcb,
        xlim = c(Min.pc1, Max.pc1), ylim = c(Min.pc2, Max.pc2),
        main = paste("PCA Loading Plot (", scaling, ")", sep = ""),
        sub = paste("Cumulative Proportion of Variance Explained = ",
            cum, "%", sep = ""))
    axis(1, at = Mpc1, pos = c(0, 0), labels = FALSE, col = "grey",
        lwd = 0.7)
    axis(2, at = Mpc2, pos = c(0, 0), labels = FALSE, col = "grey",
        lwd = 0.7)
    text(Loading.x[, pcx], Loading.x[, pcy], labels = rownames(Loading.x),
        cex = 0.6, pos = 1)
    E = paste(getwd(), "/PCA_Data_", scaling, "/LoadingPlot_PC",
        pcx, "vsPC", pcy, ".pdf", sep = "")
    dev.copy2pdf(file = E)
  }
```

`Plot.pca.pvalues` *PCA Loading plot color-coded for univariate significance*

Description

After the 'univariate' function has been performed, p-values of statistical significance for each variable are generated. These values are used to color the variables in a PCA loading plot according to their significance (p<0.05).

Usage

```
Plot.pca.pvalues(pcx, pcy, scaling)
```

Arguments

pcx	an integer indicating the principal component to be plotted in x
pcy	an integer indicating the principal component to be plotted in y
scaling	a character string indicating the name of the scaling previously specified in the function 'explore.data'

Details

A PCA Loading plot is graphically visualized and written in the working directory.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (pcx, pcy, scaling)
{
    loadi = paste(getwd(), "/PCA_Data_", scaling, "/PCA_LoadingsMatrix.csv",
        sep = "")
    Loading <- read.csv(loadi, sep = ",", header = TRUE)
    Loading.x <- Loading[, 2:ncol(Loading)]
    rownames(Loading.x) <- Loading[, 1]
    load.xlab = paste("Loading PC", pcx)
    load.ylab = paste("Loading PC", pcy)
    lim.load = c()
    Max.pc1 = 1.1 * (max(Loading.x[, pcx]))
    Min.pc1 = 1.1 * (min(Loading.x[, pcy]))
    Mpc1 = c(Min.pc1 * 2, Max.pc1 * 2)
    Max.pc2 = 1.1 * (max(Loading.x[, pcx]))
    Min.pc2 = 1.1 * (min(Loading.x[, pcy]))
    Mpc2 = c(Min.pc2 * 2, Max.pc2 * 2)
    colcool = "Colors_Pvalues"
    pwdcol = paste(getwd(), "/Univariate/Pvalues/", colcool,
        sep = "")
    col.pv = read.csv(pwdcol, header = TRUE)
    col.pv = matrix(col.pv[, -1], ncol = 1)
```

```
  dev.new()
  plot(Loading.x[, pcx], Loading.x[, pcy], col = col.pv, xlab = load.xlab,
      ylab = load.ylab, xlim = c(Min.pc1, Max.pc1), ylim = c(Min.pc2,
          Max.pc2), main = "PCA - Loading Plot (Significance-colored variables)",
      sub = "Variables in red showed Pvalue < 0.05")
  axis(1, at = Mpc1, pos = c(0, 0), labels = FALSE, col = "grey",
      lwd = 0.7)
  axis(2, at = Mpc2, pos = c(0, 0), labels = FALSE, col = "grey",
      lwd = 0.7)
  text(Loading.x[, pcx], Loading.x[, pcy], labels = rownames(Loading.x),
      cex = 0.6, pos = 1)
  E = paste(getwd(), "/PCA_Data_", scaling, "Loading_PC", pcx,
      "vsPC", pcy, "_Pvalues-colored.pdf", sep = "")
  dev.copy2pdf(file = E)
}
```

`Plot.pca.score` *Plot PCA Score Plot*

Description

This function is implemented in the unique function 'plot.pca'. It generated and visualizes PCA score plot color-coded according with the class definition in the second column of the file. This function cannot be used without having previously used the function 'explore.data'.

Usage

```
Plot.pca.score(pcx, pcy, scaling)
```

Arguments

`pcx`	an integer indicating the principal component to be plotted in x.
`pcy`	an integer indicating the principal component to be plotted in y
`scaling`	a character string indicating the name of the scaling previously specified in the function 'explore.data'

Details

For details see ?plot.pca.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (pcx, pcy, scaling)
{
    score = paste(getwd(), "/PCA_Data_", scaling, "/PCA_ScoreMatrix.csv",
        sep = "")
    ppppp = paste(getwd(), "/PCA_Data_", scaling, "/PCA_P", sep = "")
    Score <- read.csv(score, sep = ",", header = TRUE)
    Score.x <- Score[, 2:ncol(Score)]
    rownames(Score.x) <- Score[, 1]
    pwdK = paste(getwd(), "/Preprocessing_Data_", scaling, "/class.csv",
        sep = "")
    k = read.csv(pwdK)
    Pvar <- read.csv(ppppp, sep = ",", header = TRUE)
    Pvar.x <- Pvar[, 2:ncol(Pvar)]
    rownames(Pvar.x) <- Pvar[, 1]
    pca <- paste("PC", pcx, " (", Pvar[pcx, 2], ") %")
    pcb <- paste("PC", pcy, " (", Pvar[pcy, 2], ")%")
    cum = Pvar[pcx, 2] + Pvar[pcy, 2]
    xlab = c(pca)
    ylab = c(pcb)
    lim = c()
    max.pc1 = 1.3 * (max(abs(Score.x[, pcx])))
    max.pc2 = 1.3 * (max(abs(Score.x[, pcy])))
    if (max.pc1 > max.pc2) {
        lim = c(-max.pc1, max.pc1)
    }
    else {
        lim = c(-max.pc2, max.pc2)
    }
    tutticolors = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, "rosybrown4",
        "green4", "navy", "purple2", "orange", "pink", "chocolate2",
        "coral3", "khaki3", "thistle", "turquoise3", "palegreen1",
        "moccasin", "olivedrab3", "azure4", "gold3", "deeppink"),
        ncol = 1)
    col = c()
    for (i in 1:nrow(k)) {
        col = c(col, tutticolors[k[i, 2], ])
    }
    dev.new()
    plot(Score.x[, pcx], Score.x[, pcy], col = col, xlab = xlab,
      ylab = ylab, xlim = lim, ylim = lim, pch = 19, sub = paste("Cumulative Proportion of Variance Explained = ",
            cum, "%", sep = ""), main = paste("PCA Score Plot (",
            scaling, ")", sep = ""))
    axis(1, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
        lwd = 0.7)
    axis(2, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
        lwd = 0.7)
    library(car)
    dataEllipse(Score.x[, pcx], Score.x[, pcy], levels = c(0.95),
```

```
        add = TRUE, col = "black", lwd = 0.4, plot.points = FALSE,
        center.cex = 0.2)
    text(Score.x[, pcx], Score.x[, pcy], col = col, cex = 0.5,
        labels = rownames(Score.x), pos = 1)
    D <- paste(getwd(), "/PCA_Data_", scaling, "/ScorePlot_PC",
        pcx, "vsPC", pcy, ".pdf", sep = "")
    dev.copy2pdf(file = D)
  }
```

Plot.plsda *PLS plots*

Description

Visualizes and write PLS score and w*c plots.

Usage

```
Plot.plsda(pcx, pcy, scaling)
```

Arguments

pcx	an integer indicating the principal component to be plotted in x
pcy	an integer indicating the principal component to be plotted in y
scaling	a character string indicating the name of the scaling previously specified in the function 'explore.data'

Details

Score and w*c plots are graphically visualized and written in the directory 'PLS-DA'.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (pcx, pcy, scaling)
{
    pwd.score = paste(getwd(), "/PLS-DA_", scaling, "/PLSDA_Scores_",
        scaling, ".csv", sep = "")
    score = read.csv(pwd.score, header = TRUE)
    score.x = score[, -1]
    rownames(score.x) = score[, 1]
    pwd.load = paste(getwd(), "/PLS-DA_", scaling, "/PLSDA_Loadings_",
        scaling, ".csv", sep = "")
```

```
loading = read.csv(pwd.load, header = TRUE)
loading.x = loading[, -1]
rownames(loading.x) = loading[, 1]
pwd.p = paste(getwd(), "/PLS-DA_", scaling, "/PLSDA_P_",
    scaling, ".csv", sep = "")
p = read.csv(pwd.p, header = TRUE)
p.x = matrix(p[, -1], ncol = 1)
pwd.ptot = paste(getwd(), "/PLS-DA_", scaling, "/PLSDA_Ptot_",
    scaling, ".csv", sep = "")
p = read.csv(pwd.ptot, header = TRUE)
pvar.a = p.x[pcx, ]/p
pvar.b = p.x[pcy, ]/p
pvar.ai = round(pvar.a * 100, 1)
pvar.bi = round(pvar.b * 100, 1)
cum = pvar.ai + pvar.bi
xlab = paste("Component", pcx, "(", pvar.ai, "%)", sep = "")
ylab = paste("Component", pcy, "(", pvar.bi, "%)", sep = "")
max.pc1 = 1.3 * (max(abs(score.x[, pcx])))
max.pc2 = 1.3 * (max(abs(score.x[, pcy])))
lim = c()
if (max.pc1 > max.pc2) {
    lim = c(-max.pc1, max.pc1)
}
else {
    lim = c(-max.pc2, max.pc2)
}
pwdK = paste(getwd(), "/Preprocessing_Data_", scaling, "/class.csv",
    sep = "")
k = read.csv(pwdK, header = TRUE)
k.s = k[order(k[, 2]), ]
tutticolors = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, "rosybrown4",
    "green4", "navy", "purple2", "orange", "pink", "chocolate2",
    "coral3", "khaki3", "thistle", "turquoise3", "palegreen1",
    "moccasin", "olivedrab3", "azure4", "gold3", "deeppink"),
    ncol = 1)
col = c()
for (i in 1:nrow(k.s)) {
    col = c(col, tutticolors[k.s[i, 2], ])
}
plot(score.x[, pcx], score.x[, pcy], col = col, pch = 19,
    xlab = c(xlab), ylab = c(ylab), xlim = lim, ylim = lim,
    sub = paste("Cumulative Proportion of Variance Explained = ",
        cum, "%", sep = ""), main = paste("PLS-DA Score Plot (",
        scaling, ")", sep = ""))
text(score.x[, pcx], score.x[, pcy], col = col, cex = 0.5,
    labels = rownames(score.x), pos = 1)
axis(1, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
axis(2, at = lim * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
library(car)
dataEllipse(score.x[, pcx], score.x[, pcy], levels = c(0.95),
    add = TRUE, col = "black", lwd = 0.4, plot.points = FALSE,
```

```
    center.cex = 0.2)
  dirout = paste(getwd(), "/PLS-DA_", scaling, "/", sep = "")
  scor = paste(dirout, "ScorePlot_PLS-DA_", scaling, ".pdf",
    sep = "")
  dev.copy2pdf(file = scor)
  Max.pc1 = 1.1 * (max(loading.x[, pcx]))
  Min.pc1 = 1.1 * (min(loading.x[, pcx]))
  Mpc1 = c(Min.pc1, Max.pc1)
  Max.pc2 = 1.1 * (max(loading.x[, pcy]))
  Min.pc2 = 1.1 * (min(loading.x[, pcy]))
  Mpc2 = c(Min.pc2, Max.pc2)
  dev.new()
  plot(loading.x[, pcx], loading.x[, pcy], xlim = Mpc1, ylim = Mpc2,
    xlab = paste("w*c values ", pcx, sep = ""), ylab = paste("w*c values ",
      pcy, sep = ""), main = paste("PLS-DA Loading Plot (",
      scaling, ")", sep = ""))
  text(loading.x[, pcx], loading.x[, pcy], labels = rownames(loading.x),
    cex = 0.7, pos = 1)
  axis(1, at = Mpc1 * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
  axis(2, at = Mpc2 * 2, pos = c(0, 0), labels = FALSE, col = "grey",
    lwd = 0.7)
  load = paste(dirout, "W*cPlot_PLS-DA_", scaling, ".pdf",
    sep = "")
  dev.copy2pdf(file = load)
}
```

---

plsda *Partial Least Square Discriminant Analysis*

Description

Perform PLS-DA according to the class subgroups.

Usage

```
plsda(scaling)
```

Arguments

scaling	a character string indicating the name of the scaling previously specified in the function 'explore.data'

Details

The number of components to be calculated are defined as the number of classes - 1. A plot reporting pairwise representation of the components is graphically visualized and written in the directory 'PLS-DA', together with the PLS score and loading matrices.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (scaling)
{
    pwd.x = paste(getwd(), "/Preprocessing_Data_", scaling, "/ProcessedTable.csv",
        sep = "")
    x = read.csv(pwd.x, header = TRUE)
    x.x = x[, 2:ncol(x)]
    rownames(x.x) = x[, 1]
    pwdK = paste(getwd(), "/Preprocessing_Data_", scaling, "/class.csv",
        sep = "")
    k = read.csv(pwdK, header = TRUE)
    k.x = matrix(k[, -1], ncol = 1)
    x.n = cbind(k.x, x.x)
    sorted = x.n[order(x.n[, 1]), ]
    g = c()
    for (i in 1:nrow(sorted)) {
        if (any(g == sorted[i, 1])) {
            g = g
        }
        else {
            g = matrix(c(g, sorted[i, 1]), ncol = 1)
        }
    }
    dimB = nrow(g) * nrow(sorted)
    B = matrix(rep(NA, dimB), ncol = nrow(g))
    for (i in 1:nrow(sorted)) {
        for (j in 1:nrow(g)) {
            if (sorted[i, 1] == j) {
                B[i, j] = 1
            }
            else {
                B[i, j] = 0
            }
        }
    }
    library(pls)
    sorted.x = sorted[, -1]
    sorted.un = matrix(unlist(sorted.x), ncol = ncol(sorted.x))
    P = plsr(B ~ sorted.un, ncomp = nrow(g) - 1, method = c("kernelpls"),
        validation = "CV")
    rownames(P$scores) = rownames(sorted.x)
    rownames(P$loadings) = colnames(sorted.x)
    dirout = paste(getwd(), "/PLS-DA_", scaling, "/", sep = "")
    dir.create(dirout)
    out.score = paste(dirout, "PLSDA_Scores_", scaling, ".csv",
```

```
    sep = "")
write.csv(P$scores, out.score)
out.load = paste(dirout, "PLSDA_Loadings_", scaling, ".csv",
    sep = "")
write.csv(P$loadings, out.load)
k = matrix(sorted[, 1], ncol = 1)
tutticolors = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, "rosybrown4",
    "green4", "navy", "purple2", "orange", "pink", "chocolate2",
    "coral3", "khaki3", "thistle", "turquoise3", "palegreen1",
    "moccasin", "olivedrab3", "azure4", "gold3", "deeppink"),
    ncol = 1)
col = c()
for (i in 1:nrow(k)) {
    col = c(col, tutticolors[k[i, ], ])
}
if (ncol(P$scores) == 1) {
    xlab = "Samples"
    ylab = "Score values Component 1"
    plot(P$scores[, 1], col = col, pch = 19, xlab = c(xlab),
        ylab = c(ylab), main = paste("PLS-DA Score Plot (",
            scaling, ")", sep = ""))
    lim1 = nrow(P$scores) * 2
    axis(1, at = c(-lim1, lim1), col = "grey", pos = c(0,
        0), labels = FALSE, lwd = 1)
    text(P$scores[, 1], col = col, cex = 0.5, pos = 1, labels = rownames(P$scores))
    pwdout = paste(dirout, "ScorePlot_PLSDA_1Component_",
        scaling, ".pdf", sep = "")
    dev.copy2pdf(file = pwdout)
    Max.pc2 = 1.1 * (max(P$loadings[, 1]))
    Min.pc2 = 1.1 * (min(P$loadings[, 1]))
    Mpc2 = c(Min.pc2, Max.pc2)
    dev.new()
    plot(P$loadings[, 1], ylim = Mpc2, main = paste("PLS-DA Loading Plot (",
       scaling, ")", sep = ""), xlab = "Variables", ylab = "W*c values (Component1)")
    text(P$loadings[, 1], cex = 0.7, pos = 1, labels = rownames(P$loadings))
    pwdout1 = paste(dirout, "W*cPlot_PLSDA_1Component_",
        scaling, ".pdf", sep = "")
    dev.copy2pdf(file = pwdout1)
}
else {
    pairs = c()
    if (ncol(P$scores) >= 10) {
        pairs = c(10)
    }
    else {
        pairs = c(ncol(P$scores))
    }
    pairs(P$scores[, 1:pairs], col = col)
    pairs = paste(dirout, "Pairs_PLSDA_", scaling, ".pdf",
        sep = "")
    dev.copy2pdf(file = pairs)
}
p.v = matrix(P$Xvar, ncol = 1)
```

```
  p.v.csv = paste(dirout, "PLSDA_P_", scaling, ".csv", sep = "")
  write.csv(p.v, file = p.v.csv)
  p.vtot = matrix(P$Xtotvar, ncol = 1)
  p.vtot.csv = paste(dirout, "PLSDA_Ptot_", scaling, ".csv",
      sep = "")
  write.csv(p.vtot, file = p.vtot.csv, row.names = FALSE)
}
```

`pvalues` *P-values reporting*

Description

This function is implemented in the unique function for univariate statistical analysis 'univariate'. This function reports final p-values foe each pairwise caomparison of sample groups, by merging the p-values from Welch's T test for normally distributed variables with p-values from Wilcoxon-Mann Whitney U test for not-normally distributed variables.

Usage

```
pvalues(file, mtc)
```

Arguments

`file`	a connection or a character string giving the name of the file containing the variables
`mtc`	logical, default is 'TRUE'. Perform multiple testing correction with Banjamini-Hochberg method.

Details

A directory called 'Pvalues' is automatically generated in the directory 'Univariate' in the working directory. A report with merged p-values from each test is written.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (file, mtc)
{
    pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
    file = pwdfile
    x <- read.csv(file, sep = ",", header = TRUE)
    x.x = x[, 3:ncol(x)]
```

```
rownames(x.x) = x[, 2]
k = matrix(x[, 1], ncol = 1)
x.n = cbind(k, x.x)
sorted = x.n[order(x.n[, 1]), ]
sorted.x = sorted[, -1]
g = c()
for (i in 1:nrow(sorted)) {
    if (any(g == sorted[i, 1])) {
        g = g
    }
    else {
        g = matrix(c(g, sorted[i, 1]), ncol = 1)
    }
}
NoF = nrow(g)
dirout.pv = paste(getwd(), "/Univariate/Pvalues/", sep = "")
dir.create(dirout.pv)
dirout.sign = paste(getwd(), "/Univariate/Significant_Variables/",
    sep = "")
dir.create(dirout.sign)
for (i in 1:NoF) {
    for (j in 1:NoF) {
        if (i < j) {
            shapi = paste("ShapiroTest.", i, ".csv", sep = "")
            shapj = paste("ShapiroTest.", j, ".csv", sep = "")
            shap.pwdi = paste(getwd(), "/Univariate/Shapiro_Tests/",
              shapi, sep = "")
            shap.pwdj = paste(getwd(), "/Univariate/Shapiro_Tests/",
              shapj, sep = "")
            Si = read.csv(shap.pwdi, header = TRUE)
            Sj = read.csv(shap.pwdj, header = TRUE)
            Si = matrix(Si[, -1], ncol = 1)
            Sj = matrix(Sj[, -1], ncol = 1)
            welchij = paste("WelchTest_", i, "vs", j, ".csv",
              sep = "")
            welch.pwdij = paste(getwd(), "/Univariate/WelchTest/",
              welchij, sep = "")
            Wij = read.csv(welch.pwdij, header = TRUE)
            Wij = matrix(Wij[, -1], ncol = 1)
            wmwp = paste("WMWTest_pvalues_", i, "vs", j,
              ".csv", sep = "")
            wmw.pwd = paste(getwd(), "/Univariate/Mann-Whitney_Tests/",
              wmwp, sep = "")
            WMWp = read.csv(wmw.pwd, header = TRUE)
            WMWp = matrix(WMWp[, -1], ncol = 1)
            fin = ncol(sorted) - 1
            pvalues = matrix(rep(1, fin))
            for (q in 1:fin) {
              if (Si[q, ] > 0.05 & Sj[q, ] > 0.05) {
                pvalues[q, ] <- Wij[q, ]
              }
              else {
                pvalues[q, ] <- WMWp[q, ]
```

```
                }
              }
              if (mtc) {
                pval.corr = matrix(p.adjust(pvalues, method = c("BH"),
                  n = nrow(pvalues)), ncol = 1)
                rownames(pval.corr) = colnames(x.x)
                pvalfin = paste("Pvalues_", i, "vs", j, ".csv",
                  sep = "")
                assign(pvalfin, pval.corr)
                write.csv(pval.corr, paste(dirout.pv, pvalfin,
                  sep = ""))
                sign = c()
                for (q in 1:fin) {
                  if (pval.corr[q, ] < 0.05) {
                    sign = matrix(c(sign, colnames(sorted.x)[q]))
                  }
                }
                signnam = paste("Significant_Variables_", i,
                  "vs", j, ".csv", sep = "")
                assign(signnam, sign)
                write.csv(sign, paste(dirout.sign, signnam,
                  sep = ""), row.names = FALSE)
              }
              else {
                rownames(pvalues) = colnames(x.x)
                pvalfin = paste("Pvalues_", i, "vs", j, ".csv",
                  sep = "")
                assign(pvalfin, pvalues)
                write.csv(pvalues, paste(dirout.pv, pvalfin,
                  sep = ""))
                sign = c()
                for (q in 1:fin) {
                  if (pvalues[q, ] < 0.05) {
                    sign = matrix(c(sign, colnames(sorted.x)[q]))
                  }
                }
                signnam = paste("Significant_Variables_", i,
                  "vs", j, ".csv", sep = "")
                assign(signnam, sign)
                write.csv(sign, paste(dirout.sign, signnam,
                  sep = ""), row.names = FALSE)
              }
            }
        }
    }
}
```

---

ransy *Ratio Analysis NMR SpectroscopY*

Description

Starting from a specified variable (matrix column) called 'driver', it calculates the ratio between this variable and all the other variables in the dataset, returning a barplot indicating the ratio values calculated between the driver and the other variables. This function may help with NMR molecular identification and assignment.

Usage

```
ransy(scaling, driver.peak)
```

Arguments

`scaling`	a character string indicating the name of the scaling previously used with the function 'explore.data'
`driver.peak`	An integer indicating the column number of the dataset to use as 'driver peak'.

Details

'driver.peak' can be chosen through the function 'chose.driver' (see ?chose.driver), which lists all the variables (column names) present in the dataset.

A barplot is generated color-coded for the ratio values calculated between the driver and the other variables. This plot is written within the directory 'RANSY', in the working directory.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

References

Wei, S. et al. Ratio Analysis Nuclear Magnetic Resonance Spectroscopy for Selective Metabolite Identification in Complex Samples. (2011) Anal Chem 83(20):7616-7623.

See Also

chose.driver, stocsy.1d

Examples

```
## The function is currently defined as
function (scaling, driver.peak)
{
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling, "/ProcessedTable.csv",
        sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.t <- x.x
    one = matrix(rep(1, ncol(x.t)), nrow = 1)
```

```
  driver = x.t[, driver.peak] %*% one
  D = x.t/driver
  m = matrix(colMeans(D), nrow = 1)
  sd = matrix(apply(D, 2, sd), nrow = 1)
  R = m/sd
  R[, driver.peak] = 1
  R[, driver.peak] = max(R)
  Rt = t(R)
  library(gplots)
  plot(Rt, type = "h", main = paste("RANSY (", rownames(x.x)[driver.peak],
      ")", sep = ""), ylab = paste("Mean/sd of ratio with ",
      rownames(x.x)[driver.peak], sep = ""), xlab = "Variables")
  text(Rt, labels = colnames(x.x), cex = 0.6)
  dirout = paste(getwd(), "/RANSY/", sep = "")
  dir.create(dirout)
  out = paste(dirout, "ransy_", colnames(x.x)[driver.peak],
      ".pdf", sep = "")
  dev.copy2pdf(file = out)
}
```

---

shapiro *Shapiro Wilk's test*

Description

This function is implemented in the unique function for univariate statistical analysis 'univariate'. Performs Shapiro Wilk's test for normality on the dataset according to the class definition provided in the second column of the file. For details see 'univariate'.

Usage

```
shapiro(file)
```

Arguments

file	a connection or a character string giving the name of the file containing the variables (matrix columns) to test.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (file)
{
```

```
  pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
  file = pwdfile
  x <- read.csv(file, sep = ",", header = TRUE)
  x.x = x[, 3:ncol(x)]
  rownames(x.x) = x[, 2]
  k = matrix(x[, 1], ncol = 1)
  x.n = cbind(k, x.x)
  sorted = x.n[order(x.n[, 1]), ]
  g = c()
  for (i in 1:nrow(sorted)) {
      if (any(g == sorted[i, 1])) {
          g = g
      }
      else {
          g = matrix(c(g, sorted[i, 1]), ncol = 1)
      }
  }
  dirout.r = paste(getwd(), "/Univariate/Groups", sep = "")
  dir.create(dirout.r)
  dirout.s = paste(getwd(), "/Univariate/Shapiro_Tests", sep = "")
  dir.create(dirout.s)
  for (i in 1:nrow(g)) {
      vuota <- c()
      fin = matrix(rep(NA, ncol(sorted)), nrow = 1)
      for (j in 1:nrow(sorted)) {
          if (sorted[j, 1] == i) {
              vuota <- matrix(sorted[j, ], nrow = 1)
              rownames(vuota) = rownames(sorted)[j]
              fin = rbind(fin, vuota)
          }
      }
      nam = paste("r", i, sep = ".")
      n = matrix(fin[-1, ], ncol = ncol(sorted))
      n.x = matrix(n[, -1], ncol = ncol(sorted) - 1)
      lastcol = ncol(sorted)
      n.x = n.x[, -lastcol]
      colnames(n.x) = colnames(x.x)
      name = as.matrix(assign(nam, n.x))
      shapname = paste("shapiro", i, sep = ".")
      shapiro = matrix(rep(NA, ncol(n.x)))
      for (q in 1:ncol(name)) {
          notAlist = c()
          notAlist = matrix(unlist(name[, q]))
          shapiro[q, ] = shapiro.test(notAlist)$p.value
          assign(shapname, shapiro)
      }
      outputfile = paste("r.", i, ".csv", sep = "")
      write.csv(name, paste(dirout.r, outputfile, sep = "/"))
      outshapiro = paste("ShapiroTest.", i, ".csv", sep = "")
      shapiro[is.na(shapiro)] = 1
      write.csv(shapiro, paste(dirout.s, outshapiro, sep = "/"))
  }
}
```

stocsy *Statistical TOtal Correlation SpectroscopY*

Description

Function to calculate correlation matrix and plot corresponding color-coded heatmap, useful for NMR molecular identification and assignment.

Usage

```
stocsy(scaling, threshold = FALSE, pos.threshold, neg.threshold)
```

Arguments

scaling	a character string indicating the name of the scaling previously used with the function 'explore.data'.
threshold	logical. The possibility to indicate or not a positive and negative threshold regulating the heatmap coloring.
pos.threshold	a decimal (or integer) indicating the upper level beyond which correlation values are visualized.
neg.threshold	a decimal (or integer) indicating the lower level below which correlation values are visualized.

Details

'threshold' is 'FALSE' by default. If 'TRUE' positive and negative threshold values must be specified. In this case two plots are created, one with no color thresholds and one with the indicated thresholds; both these heatmaps are written in the directory 'STOCSY', automatically created in the working directory.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

References

Cloarec, O et al. Statistical total correlation spectroscopy: an exploratory approach for latent biomarker identification from metabolic 1H NMR data sets. (2005) Anal Chem. 77(5):1282-9.

Examples

```
## The function is currently defined as
function (scaling, threshold = FALSE, pos.threshold, neg.threshold)
{
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling, "/ProcessedTable.csv",
```

```
        sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.t <- x.x
    mycor = cor(x.t, method = c("pearson"))
    library(gplots)
    col = colorpanel(50, "blue", "white", "red")
    image(mycor, axes = FALSE, col = col, main = "STOCSY")
   axis(side = 1, labels = colnames(mycor), at = seq(0, 1, length = length(colnames(mycor))),
        las = 2, cex.axis = 0.7)
   axis(side = 2, labels = colnames(mycor), at = seq(0, 1, length = length(colnames(mycor))),
        las = 2, cex.axis = 0.7)
    dirout = paste(getwd(), "/STOCSY/", sep = "")
    dir.create(dirout)
    o = paste(dirout, "STOCSY.pdf", sep = "")
    dev.copy2pdf(file = o)
    o.cor = paste(dirout, "CorrelationMatrix.csv", sep = "")
    write.csv(mycor, file = o.cor)
    if (threshold) {
        dev.new()
        image(mycor, axes = FALSE, zlim = c(pos.threshold, 1),
            col = "red", main = paste("STOCSY <", neg.threshold,
                " & >", pos.threshold, sep = ""))
        image(mycor, axes = FALSE, zlim = c(-1, neg.threshold),
            col = "navy", add = TRUE)
        axis(side = 1, labels = colnames(mycor), at = seq(0,
            1, length = length(colnames(mycor))), las = 2, cex.axis = 0.7)
        axis(side = 2, labels = colnames(mycor), at = seq(0,
            1, length = length(colnames(mycor))), las = 2, cex.axis = 0.7)
        out = paste(dirout, "STOCSY_", pos.threshold, "_", neg.threshold,
            ".pdf", sep = "")
        dev.copy2pdf(file = out)
    }
}
```

`stocsy.1d` *Statistical TOtal Correlation SpectroscopY - monodimensional*

Description

Calculate correlations of a specified variable with all the other variables in a dataset and generate the corresponding barplot indicating each correlation value.

Usage

```
stocsy.1d(scaling, driver.peak)
```

Arguments

scaling	a character string indicating the name of the scaling previously used with the function 'explore.data'
driver.peak	An integer indicating the column number of the dataset to use as 'driver peak'.

Details

A barplot color-coded for positive (red), negative (blue) and null (green) correlations is generated and written in the directory 'STOCSY_1D'.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

References

Cloarec, O et al. Statistical total correlation spectroscopy: an exploratory approach for latent biomarker identification from metabolic 1H NMR data sets. (2005) Anal Chem. 77(5):1282-9.

Examples

```
## The function is currently defined as
function (scaling, driver.peak)
{
    pwd.n = paste(getwd(), "/Preprocessing_Data_", scaling, "/ProcessedTable.csv",
        sep = "")
    x <- read.csv(pwd.n, sep = ",", header = TRUE)
    x.x <- x[, 2:ncol(x)]
    rownames(x.x) <- x[, 1]
    x.t <- x.x
    mycor = cor(x.t, method = c("pearson"))
    library(gplots)
    pal = matrix(rich.colors(41), nrow = 1)
    sec = matrix(seq(-1, 1, 0.05), nrow = 1)
    d = matrix(mycor[, driver.peak], ncol = 1)
    sec40 = ncol(sec) - 1
    col = c()
    for (i in 1:nrow(d)) {
        for (j in 1:sec40) {
            if (sec[, j] <= d[i, ] & d[i, ] <= sec[, j + 1]) {
                col = matrix(c(col, pal[, j]), ncol = 1)
            }
        }
    }
    plot(mycor[, driver.peak], type = "h", col = col, xlab = "Variables",
        ylab = paste("Coefficient of correlation with ", rownames(x.x)[driver.peak],
            sep = ""), main = paste("STOCSY 1D (", rownames(x.x)[driver.peak],
            ")", sep = ""), ylim = c(-1, 1))
```

```
  text(mycor[, driver.peak], labels = colnames(x.x), cex = 0.5,
      col = col)
  dirout = paste(getwd(), "/STOCSY_1D/", sep = "")
  dir.create(dirout)
  out = paste(dirout, "stocsy_1d_", colnames(x.x)[driver.peak],
      ".pdf", sep = "")
  dev.copy2pdf(file = out)
}
```

`univariate` *Univariate statistical analysis*

Description

Function to perform univariate statistical analysis with parametric and non parametric testings. A decisional tree algorithm is implemented, assessing whether a variable is normally distributed or not (Shapiro Wilk's test) and performing Welch's T test or Wilcoxon-Mann Whitney U test, depending on normality.

Usage

```
univariate(file, imputation = FALSE, imput, normalize = TRUE, multi.test = TRUE, plot.volcano = FALSE,
```

Arguments

`file`	a connection or a character string giving the name of the file containing the variables (matrix columns) to test.
`imputation`	logical, whether to perform imputation of missing values. Default is 'FALSE'.
`imput`	character vector indicating the type of imput that should be used to imput missing values. See details for imput options.
`normalize`	logical, whether to perform normalization. Default is 'TRUE'.
`multi.test`	logical, default is 'TRUE'. Performs correction for multiple testing with Benjamini-Hochberg method.
`plot.volcano`	logical, default is 'FALSE', visualizes volcano plots of each pairwise comparison between groups.
`save.boxplot`	logical, default is 'TRUE', plots and saves boxplot.

Details

The 'file' provided has to be a matrix in .csv form, formatted with the first column indicating the name of the samples and the second column indicating the class of belonging of each samples (e.g. treatment groups, healthy/diseased, ...). The header of the matrix must contain the name of each variable in the dataset.

A directory called 'Univariate' is automatically generated in the working directory. In this directory are stored all the results from the function 'univariate'.

There for options for imputing missing values: "mean", "minimum", "half.minimum", "zero". For specifying the type of imput to be used, the field 'imputation' must be turned to 'TRUE'.

If 'normalize' is 'TRUE', a normalization is performed on total intensity, i.e. the sum of all variables for each sample is calculated and used as normalizing factor for each variable.

Volcano plots are generated automatically and written in the directory 'Volcano_Plots', within the 'Univariate' directory. If 'plot.volcano' is 'TRUE' such plots are also visualized. Moreover, fold changes and p-values calculated for creating volcano plots are written in the directory 'Fold_Changes' and in the directory 'Pvalues' respectively, for each pairwise comparison between groups.

The function is 'univariate' is a hub of univariate statistical tests. First, Shapiro Wilk's test for normality is performed for each variable, relying on class definition provided in the second column of the 'file'. If a variable results normally distributed, then Welch's T test is performed; otherwise, if a variable is not normally distributed, Wilcoxon-Mann Whitney U test is performed. For both these tests a threshold of 0.05 is used for assessing significance. After these testings a directory called 'Shapiro_Tests' is created containing the scores of every variable for normality; a directory called 'Welch_Tests' is created containing the p-values for each variable tested, for each pairwise comparison between gropus of samples; a directory called 'Mann-Whitney_Tests' is created containing the p-values for each variable tested, for each pairwise comparison between gropus of samples; a directory called 'Significant_Variables' is created reporting only the variables showing a p-value < 0.05, hence considered significantly different between groups.

Boxplots for each variable, plotted according to class definition provided in the second column of the 'file', are generated and written in the directory called 'BoxPlot', automatically generated in the working directory.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

References

Goodpaster, AM, et al. Statistical Significance analysis of nuclear magnetic resonance-based metabonomics data. (2010) Anal Biochem. 401:134-143.

Examples

```
## The function is currently defined as
function (file, imputation = FALSE, imput, normalize = TRUE,
    multi.test = TRUE, plot.volcano = FALSE, save.boxplot = TRUE)
{
    dirout.uni = paste(getwd(), "/Univariate/", sep = "")
    dir.create(dirout.uni)
    comp = read.csv(file, sep = ",", header = TRUE)
    comp.x = comp[, 3:ncol(comp)]
    for (i in 1:nrow(comp.x)) {
        for (j in 1:ncol(comp.x)) {
            if (comp.x[i, j] <= 0) {
                comp.x[i, j] = runif(1, 0, 1e-10)
```

```
        }
    }
}
comp.x = cbind(comp[, 2], comp[, 1], comp.x)
pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
write.csv(comp.x, pwdfile, row.names = FALSE)
if (imputation) {
    x <- read.csv(pwdfile, sep = ",", header = TRUE)
    x.x <- x[, 3:ncol(x)]
    rownames(x.x) <- x[, 2]
    y = x.x
    r = is.na(y)
    for (k in 1:ncol(r)) {
        vec = matrix(r[, k], ncol = 1)
        who.miss.rows = which(apply(vec, 1, function(i) {
            any(i)
        }))
        if (length(who.miss.rows) > nrow(y) * 0.8) {
            warning(paste("The variable -", colnames(y)[k],
             "- has a number of missing values > 80%, therefore has been eliminated",
              sep = " "))
            y = y[, -k]
        }
    }
    r = is.na(y)
    who.miss.columns = c()
    for (i in 1:nrow(y)) {
        for (j in 1:ncol(y)) {
            if (r[i, j] == TRUE) {
              if (imput == "mean") {
                v2 = matrix(r[, j], ncol = 1)
                who.miss.rows = which(apply(v2, 1, function(i) {
                  any(i)
                }))
                rmax = sd(y[-who.miss.rows, j])/1000
                rmin = -sd(y[-who.miss.rows, j])/1000
                random = sample(rmin:rmax, 1, replace = TRUE)
                y[i, j] = mean(y[-who.miss.rows, j]) + random
                print(paste("Imputing missing value of variable -",
                  colnames(y)[j], "- for the observation -",
                  rownames(y)[i], "- with", imput, "value",
                  sep = " "))
              }
              else if (imput == "minimum") {
                v2 = matrix(r[, j], ncol = 1)
                who.miss.rows = which(apply(v2, 1, function(i) {
                  any(i)
                }))
                rmax = sd(y[-who.miss.rows, j])/1000
                rmin = -sd(y[-who.miss.rows, j])/1000
                random = sample(rmin:rmax, 1, replace = TRUE)
                y[i, j] = min(y[-who.miss.rows, j]) + random
                print(paste("Imputing missing value of variable -",
```

```
                    colnames(y)[j], "- for the observation -",
                    rownames(y)[i], "- with", imput, "value",
                    sep = " "))
                }
                else if (imput == "half.minimum") {
                  v2 = matrix(r[, j], ncol = 1)
                  who.miss.rows = which(apply(v2, 1, function(i) {
                    any(i)
                  }))
                  rmax = sd(y[-who.miss.rows, j])/1000
                  rmin = -sd(y[-who.miss.rows, j])/1000
                  random = sample(rmin:rmax, 1, replace = TRUE)
                  y[i, j] = (min(y[-who.miss.rows, j])/2) +
                    random
                  print(paste("Imputing missing value of variable -",
                    colnames(y)[j], "- for the observation -",
                    rownames(y)[i], "- with", imput, "value",
                    sep = " "))
                }
                else if (imput == "zero") {
                  v2 = matrix(r[, j], ncol = 1)
                  who.miss.rows = which(apply(v2, 1, function(i) {
                    any(i)
                  }))
                  rmax = sd(y[-who.miss.rows, j])/1000
                  rmin = -sd(y[-who.miss.rows, j])/1000
                  random = sample(rmin:rmax, 1, replace = TRUE)
                  y[i, j] = 0 + random
                  print(paste("Imputing missing value of variable -",
                    colnames(y)[j], "- for the observation -",
                    rownames(y)[i], "- with", imput, "value",
                    sep = " "))
                }
              }
            }
        }
        y = cbind(x[, 1], x[, 2], y)
        write.csv(y, pwdfile, row.names = FALSE)
    }
    if (normalize) {
        x <- read.csv(pwdfile, sep = ",", header = TRUE)
        x.x <- x[, 3:ncol(x)]
        x.t <- t(x.x)
        x.s <- matrix(colSums(x.t), nrow = 1)
        uni = matrix(rep(1, nrow(x.t)), ncol = 1)
        area.uni <- uni %*% x.s
        x.areanorm <- x.t/area.uni
        x.areanorm = t(x.areanorm)
        x.n = cbind(x[, 1], x.areanorm)
        rownames(x.n) = x[, 2]
        write.csv(x.n, pwdfile)
        comp <- read.csv(pwdfile, sep = ",", header = TRUE)
        comp.x = comp[, 3:ncol(comp)]
```

```
        comp.x = cbind(comp[, 2], comp[, 1], comp.x)
        write.csv(comp.x, pwdfile, row.names = FALSE)
    }
    shapiro(file)
    welch(file)
    wmw(file)
    if (multi.test) {
        pvalues(file, mtc = TRUE)
    }
    else {
        pvalues(file, mtc = FALSE)
    }
    col.pvalues(file)
    if (plot.volcano) {
        volcano(file, plot.vol = TRUE)
    }
    else {
        volcano(file, plot.vol = FALSE)
    }
    if (save.boxplot) {
        box.plot(file)
    }
}
```

`volcano` *Volcano plots*

Description

This function is implemented in the unique function for univariate statistical analysis 'univariate'. This function generates volcano plots of each variable in the data set according to the class definition provided in the second column of the file.

Usage

```
volcano(file, plot.vol)
```

Arguments

`file`	a connection or a character string giving the name of the file containing the variables (matrix columns) to plot.
`plot.vol`	logical, default is 'TRUE'. Whether to visualize or not volcano plots generated.

Details

For details see ?univariate.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (file, plot.vol)
{
    pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
    file = pwdfile
    x <- read.csv(file, sep = ",", header = TRUE)
    x.x = x[, 3:ncol(x)]
    rownames(x.x) = x[, 2]
    k = matrix(x[, 1], ncol = 1)
    x.n = cbind(k, x.x)
    sorted = x.n[order(x.n[, 1]), ]
    sorted.x = as.matrix(sorted[, -1], ncol = ncol(sorted) -
        1)
    g = c()
    for (i in 1:nrow(sorted)) {
        if (any(g == sorted[i, 1])) {
            g = g
        }
        else {
            g = matrix(c(g, sorted[i, 1]), ncol = 1)
        }
    }
    NoF = nrow(g)
    dirout.fc = paste(getwd(), "/Univariate/Fold_Changes/", sep = "")
    dir.create(dirout.fc)
    dirout.vol = paste(getwd(), "/Univariate/Volcano_Plots/",
        sep = "")
    dir.create(dirout.vol)
    for (i in 1:NoF) {
        for (j in 1:NoF) {
            if (i < j) {
                ni = paste("r.", i, ".csv", sep = "")
                nj = paste("r.", j, ".csv", sep = "")
                pwdi = paste(getwd(), "/Univariate/Groups/",
                  ni, sep = "")
                pwdj = paste(getwd(), "/Univariate/Groups/",
                  nj, sep = "")
                pv = paste(getwd(), "/Univariate/Pvalues/Pvalues_",
                  i, "vs", j, ".csv", sep = "")
                I = read.csv(pwdi, header = TRUE)
                J = read.csv(pwdj, header = TRUE)
                I = I[, -1]
                J = J[, -1]
                meanI = matrix(colMeans(I), ncol = ncol(I))
                meanJ = matrix(colMeans(J), ncol = ncol(J))
```

```
MeanI = matrix(rep(NA, ncol(I)), nrow = 1)
MeanJ = matrix(rep(NA, ncol(I)), nrow = 1)
for (m in 1:ncol(I)) {
  if (meanI[, m] < 0 | meanJ[, m] < 0) {
    MeanI[, m] = 1
    MeanJ[, m] = 1
  }
  else {
    MeanI[, m] = meanI[, m]
    MeanJ[, m] = meanJ[, m]
  }
}
FC = matrix(MeanI/MeanJ, nrow = ncol(I))
rownames(FC) = colnames(I)
fc.csvfile = paste("Fold_Change_", i, "vs", j,
  ".csv", sep = "")
write.csv(FC, paste(dirout.fc, fc.csvfile, sep = ""))
PV = read.csv(pv, header = TRUE)
PV = matrix(PV[, -1], ncol = 1)
logfc = log2(FC)
logpv = -log10(PV)
colpv = matrix(rep(NA, nrow(PV)), ncol = 1)
for (p in 1:nrow(PV)) {
  if (logfc[p, ] < -0.3219281 | logfc[p, ] >
    0.2630344) {
    if (logpv[p, ] > 1.30103) {
      colpv[p, ] = "navy"
    }
    else {
      colpv[p, ] = "dark grey"
    }
  }
  else {
    colpv[p, ] = "dark grey"
  }
}
max.fc = 1.3 * (max(abs(logfc)))
V = paste(dirout.vol, "VolcanoPlot_", i, "vs",
  j, ".pdf", sep = "")
pospv = matrix(rep(NA, nrow(PV)), ncol = 1)
for (p in 1:nrow(PV)) {
  if (logfc[p, ] < 0) {
    pospv[p, ] = 2
  }
  else {
    pospv[p, ] = 4
  }
}
pdf(V)
plot(logfc, logpv, col = colpv, pch = 19, xlim = c(-max.fc,
  max.fc), xlab = "Log2 (Fold Change)", ylab = "Log10 (Pvalue)",
  main = paste("Volcano Plot ", i, " vs ", j,
sep = ""), sub = "(Variables in Blue are significant (Pvalue<0.05) and showed Fold Changes >1.2 o
```

```
                text(logfc, logpv, labels = colnames(sorted.x),
                  cex = 0.8, pos = pospv, col = colpv)
                axis(2, at = c(-1, 150), pos = c(-0.3219281,
                  0), col = "blue", lwd = 0.3)
                axis(2, at = c(-1, 150), pos = c(0.2630344, 0),
                  col = "blue", lwd = 0.3)
                axis(1, at = c(-150, 150), pos = c(1.30103, 0),
                  col = "blue", lwd = 0.3)
                dev.off()
                if (plot.vol) {
                  dev.new()
                  plot(logfc, logpv, col = colpv, pch = 19, xlim = c(-max.fc,
                    max.fc), xlab = "Log2 (Fold Change)", ylab = "Log10 (Pvalue)",
                    main = paste("Volcano Plot ", i, " vs ",
                   j, sep = ""), sub = "(Variables in Blue are significant (Pvalue<0.05) and showed Fold Changes >1
                  text(logfc, logpv, labels = colnames(sorted.x),
                    cex = 0.8, pos = pospv, col = colpv)
                  axis(2, at = c(-1, 150), pos = c(-0.3219281,
                    0), col = "blue", lwd = 0.3)
                  axis(2, at = c(-1, 150), pos = c(0.2630344,
                    0), col = "blue", lwd = 0.3)
                  axis(1, at = c(-150, 150), pos = c(1.30103,
                    0), col = "blue", lwd = 0.3)
                }
              }
            }
          }
        }
```

`welch` *Welch's T test*

Description

This function is implemented in the unique function for univariate statistical analysis 'univariate'. This function performs Welch's T test of each variable in the data set according to the class definition provided in the second column of the file.

Usage

```
welch(file)
```

Arguments

`file`	a connection or a character string giving the name of the file containing the variables (matrix columns) to test.

Details

For details see ?univariate.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (file)
{
    pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
    file = pwdfile
    x <- read.csv(file, sep = ",", header = TRUE)
    x.x = x[, 3:ncol(x)]
    rownames(x.x) = x[, 2]
    k = matrix(x[, 1], ncol = 1)
    x.n = cbind(k, x.x)
    sorted = x.n[order(x.n[, 1]), ]
    g = c()
    for (i in 1:nrow(sorted)) {
        if (any(g == sorted[i, 1])) {
            g = g
        }
        else {
            g = matrix(c(g, sorted[i, 1]), ncol = 1)
        }
    }
    NoF = nrow(g)
    dirout.w = paste(getwd(), "/Univariate/WelchTest", sep = "")
    dir.create(dirout.w)
    for (i in 1:NoF) {
        for (j in 1:NoF) {
            if (i < j) {
                ni = paste("r.", i, ".csv", sep = "")
                nj = paste("r.", j, ".csv", sep = "")
                pwdi = paste(getwd(), "/Univariate/Groups/",
                  ni, sep = "")
                pwdj = paste(getwd(), "/Univariate/Groups/",
                  nj, sep = "")
                I = read.csv(pwdi, header = TRUE)
                J = read.csv(pwdj, header = TRUE)
                I = I[, -1]
                J = J[, -1]
                fin = ncol(sorted) - 1
                we <- matrix(rep(NA, fin))
                for (q in 1:fin) {
                  we[q, ] <- t.test(I[, q], J[, q], var.equal = F,
                    conf.level = 0.95, alternative = "two.sided")$p.value
                }
                welch.ij = paste("WelchTest_", i, "vs", j, ".csv",
                  sep = "")
                assign(welch.ij, we)
```

```
                write.csv(we, paste(dirout.w, welch.ij, sep = "/"))
            }
        }
    }
}
```

---

wmw *Wilcoxon-Mann Whitney U test*

Description

This function is implemented in the unique function for univariate statistical analysis 'univariate'. This function performs Wilcoxon-mann Whitney U test of each variable in the data set according to the class definition provided in the second column of the file.

Usage

```
wmw(file)
```

Arguments

file	a connection or a character string giving the name of the file containing the variables (matrix columns) to test.

Details

For details see ?univariate.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (file)
{
    pwdfile = paste(getwd(), "/Univariate/DataTable.csv", sep = "")
    file = pwdfile
    x <- read.csv(file, sep = ",", header = TRUE)
    x.x = x[, 3:ncol(x)]
    rownames(x.x) = x[, 2]
    k = matrix(x[, 1], ncol = 1)
    x.n = cbind(k, x.x)
    sorted = x.n[order(x.n[, 1]), ]
    g = c()
    for (i in 1:nrow(sorted)) {
```

```
        if (any(g == sorted[i, 1])) {
            g = g
        }
        else {
            g = matrix(c(g, sorted[i, 1]), ncol = 1)
        }
    }
    NoF = nrow(g)
    dirout.wm = paste(getwd(), "/Univariate/Mann-Whitney_Tests/",
        sep = "")
    dir.create(dirout.wm)
    for (i in 1:NoF) {
        for (j in 1:NoF) {
            if (i < j) {
                ni = paste("r.", i, ".csv", sep = "")
                nj = paste("r.", j, ".csv", sep = "")
                pwdi = paste(getwd(), "/Univariate/Groups/",
                  ni, sep = "")
                pwdj = paste(getwd(), "/Univariate/Groups/",
                  nj, sep = "")
                I = read.csv(pwdi, header = TRUE)
                J = read.csv(pwdj, header = TRUE)
                I = I[, -1]
                J = J[, -1]
                fin = ncol(sorted) - 1
                wilx.pv <- matrix(rep(NA, fin))
                for (q in 1:fin) {
                  wilx.pv[q, ] <- wilcox.test(I[, q], J[, q],
                    paired = FALSE, exact = NULL, correct = FALSE,
                    conf.level = 0.95, alternative = "two.sided")$p.value
                }
                wmw.ij.pv = paste("WMWTest_pvalues_", i, "vs",
                  j, ".csv", sep = "")
                assign(wmw.ij.pv, wilx.pv)
                write.csv(wilx.pv, paste(dirout.wm, wmw.ij.pv,
                  sep = ""))
            }
        }
    }
}
```

---

work.dir *Create a working directory*

Description

Generic function to create a directory in the current working directory, with copying all the files in the current directory to the newly formed one. When using the whole package 'muma' this function is recommended, as many files and directories are created.

Usage

```
work.dir(dir.name)
```

Arguments

dir.name	A character string indicating the name of the new directory.

Details

When using the package 'muma' this function should be used at the beginning, as the newly formed directory became the working directory and all files generated and written will be putted here.

Author(s)

Edoardo Gaude, Dimitrios Spiliotopoulos, Francesca Chignola, Silvia Mari, Andrea Spitaleri and Michela Ghitti

Examples

```
## The function is currently defined as
function (dir.name)
{
    WorkinDir = paste(getwd(), "/", dir.name, "/", sep = "")
    dir.create(WorkinDir)
    file = list.files()
    file.copy(file, WorkinDir)
    setwd(WorkinDir)
    unlink("muma.R")
  }
```

Index

∗Topic **\textasciitildekwd1**
box.plot, 3
chose.driver, 4
col.pvalues, 5
explore.data, 7
oplsda, 19
ostocsy, 24
outlier, 25
Plot.pca, 27
Plot.pca.loading, 28
Plot.pca.pvalues, 29
Plot.pca.score, 31
Plot.plsda, 33
plsda, 35
pvalues, 38
ransy, 40
shapiro, 42
stocsy, 44
stocsy.1d, 45
univariate, 47
volcano, 51
welch, 54
wmw, 56
work.dir, 57
∗Topic **\textasciitildekwd2**
box.plot, 3
chose.driver, 4
col.pvalues, 5
explore.data, 7
oplsda, 19
ostocsy, 24
outlier, 25
Plot.pca, 27
Plot.pca.loading, 28
Plot.pca.pvalues, 29
Plot.pca.score, 31
Plot.plsda, 33
plsda, 35
pvalues, 38
ransy, 40
shapiro, 42
stocsy, 44
stocsy.1d, 45
univariate, 47
volcano, 51
welch, 54
wmw, 56
work.dir, 57

box.plot, 3

chose.driver, 4
col.pvalues, 5

explore.data, 7

MetaBc, 16
muma *(*muma-package*)*, 2
muma-package, 2

oplsda, 19
ostocsy, 24
outlier, 25

Plot.pca, 27
Plot.pca.loading, 28
Plot.pca.pvalues, 29
Plot.pca.score, 31
Plot.plsda, 33
plsda, 35
pvalues, 38

ransy, 40

shapiro, 42
stocsy, 44
stocsy.1d, 45

univariate, 47

volcano, 51

welch, 54
wmw, 56
work.dir, 57