Package 'imputeLCMD'

February 20, 2015

Type Package

Title A collection of methods for left-censored missing data imputation

Version 2.0

Date 2015-01-18

Author Cosmin Lazar

Maintainer Cosmin Lazar <vcosminlazar@gmail.com>

Description The package contains a collection of functions for left-censored missing data imputation. Left-censoring is a special case of missing not at random (MNAR) mechanism that generates non-responses in proteomics experiments. The package also contains functions to artificially generate peptide/protein expression data (log-transformed) as random draws from a multivariate Gaussian distribution as well as a function to generate missing data (both randomly and non-randomly). For comparison reasons, the package also contains several wrapper functions for the imputation of non-responses that are missing at random. * New functionality has been added: a hybrid method that allows the imputation of missing values in a more complex scenario where the missing data are both MAR and MNAR.

License GPL (>= 2)

Depends R (>= 2.10), tmvtnorm, norm, pcaMethods, impute

NeedsCompilation no

Repository CRAN

Date/Publication 2015-01-19 07:09:10

R topics documented:

`generate.ExpressionData` *Generate Peptide/Protein Expression Data*

Description

Generates a matrix of peptide/protein expression data. It is assumed that the expression data is log - transformed. Therefore, for each sample the peptides/proteins intensities are randomly drawn from a Gaussian distribution.

Usage

```
generate.ExpressionData(nSamples1, nSamples2, meanSamples, sdSamples,
                        nFeatures, nFeaturesUp, nFeaturesDown,
                        meanDynRange, sdDynRange,
                        meanDiffAbund, sdDiffAbund)
```

Arguments

`nSamples1`	Number of samples in condition 1.
`nSamples2`	Number of samples in condition 2.
`meanSamples`	Mean value of the background noise.
`sdSamples`	Standard deviation of the background noise.
`nFeatures`	Number of peptides/proteins.
`nFeaturesUp`	Number of peptides/proteins up-regulated.
`nFeaturesDown`	Number of /peptidesproteins down-regulated.
`meanDynRange`	Mean value of the dynamic range of peptide/protein expressions.
`sdDynRange`	Standard deviation of the dynamic range of peptide/protein expressions.
`meanDiffAbund`	Mean value of the up/down-regulation.
`sdDiffAbund`	Standard deviation of the up/down-regulation.

Value

A list including elements:

`data`	Peptide/protein expression matrix
`conditions`	Vector indicating the samples in each condition
`labelFeatures`	Vector indicating features which are down/up/no regulated

Author(s)

Cosmin Lazar

See Also

`pep2prot`, `generate.RollUpMap`, `insertMVs`

Examples

```
dataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 2000, nFeaturesUp = 100, nFeaturesDown = 100,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = dataObj[[1]]

## Not run:
hist(exprsData[,1])

## End(Not run)

## The function is currently defined as
function (nSamples1, nSamples2, meanSamples, sdSamples, nFeatures,
    nFeaturesUp, nFeaturesDown, meanDynRange, sdDynRange, meanDiffAbund,
    sdDiffAbund)
{
    nSamples = nSamples1 + nSamples2
    data = matrix(rnorm(nSamples * nFeatures, meanSamples, sdSamples),
        nFeatures, nSamples)
    means = rnorm(nFeatures, meanDynRange, sdDynRange)
    data = data + means
    conditions = c(rep(1, nSamples1), rep(2, nSamples2))
    DE.coef.up = matrix(rnorm(nFeaturesUp * nSamples1, meanDiffAbund,
        sdDiffAbund), nFeaturesUp, nSamples1)
    DE.coef.down = matrix(rnorm(nFeaturesDown * nSamples2, meanDiffAbund,
        sdDiffAbund), nFeaturesDown, nSamples2)
    data[1:nFeaturesUp, conditions == 1] = DE.coef.up + data[1:nFeaturesUp,
        conditions == 1]
    data[(nFeaturesUp + 1):(nFeaturesUp + nFeaturesDown), conditions ==
        2] = DE.coef.down + data[(nFeaturesUp + 1):(nFeaturesUp +
        nFeaturesDown), conditions == 2]
    labelFeatures = c(rep(1, nFeaturesUp), rep(2, nFeaturesDown),
```

```
        rep(3, nFeatures - (nFeaturesUp + nFeaturesDown)))
  row.names(data) = 1:nFeatures
  return(list(data, conditions, labelFeatures))
}
```

`generate.RollUpMap` *Generates peptide to protein map.*

Description

Generates peptide to protein map. For a given peptide expression matrix with `nPep` peptides, this functions creates a random map to be used for the aggregation of the peptides into `nProt` proteins.

Usage

```
generate.RollUpMap(nProt, pep.Expr.Data)
```

Arguments

`nProt`	Number of proteins
`pep.Expr.Data`	Peptide expression data matrix

Value

A vector of the length `nPep` containing integer random samples in the range `[1, nProt]`

Author(s)

Cosmin Lazar

See Also

`generate.ExpressionData`, `pep2prot`

Examples

```
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]
rollUpMap = generate.RollUpMap(round(dim(exprsData)[1]/2),exprsData)

## The function is currently defined as
function (nProt, pep.Expr.Data)
{
    n = dim(pep.Expr.Data)[1]
```

```
  temp = 1:nProt
  pep.prot.Map = rep(0, n)
  pep.prot.Map[sample(temp)] = sample(temp)
  pep.prot.Map[which(pep.prot.Map == 0)] = sample.int(nProt,
      size = (n - nProt), replace = T)
  return(pep.prot.Map)
}
```

`impute.MAR` *Generic function for the imputation of MAR/MCAR missing data*

Description

Performs the imputation of missing data under the randomness assumption (either MAR or MCAR). The function allows treating the missing values using one of the following MAR/MCAR specific imputation methods: MLE, SVD, KNN.

Usage

```
impute.MAR(dataSet.mvs, model.selector, method = "MLE")
```

Arguments

`dataSet.mvs`	A data matrix containing missing values.
`model.selector`	Binary vector; "1" indicates rows that should be treated using MAR/MCAR specific methods, while "0" indicates rows treated using MNAR specific methods (the missing values corresponding to these rows are assumed to be left-censored).
`method`	The method employed for the imputation of MAR/MCAR missing data.

Value

A data matrix containing only MNAR (assumed to be left-censored) missing data.

Author(s)

Cosmin Lazar

See Also

`model.Selector`, `imp.norm`, `pca`, `impute.knn`

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# run model.Selector
m.s = model.Selector(exprsData.MD)

# perform MAR/MCAR imputation
exprsData.MAR.imputed = impute.MAR (exprsData.MD, m.s, method = "MLE")

## The function is currently defined as
function (dataSet.mvs, model.selector, method = "MLE")
{
    if (length(which(model.selector[[1]] == 1)) == 0) {
        dataSet.imputed = dataSet.mvs
    }
    else {
        dataSet.MCAR = dataSet.mvs[which(model.selector[[1]] ==
            1), ]
        switch(method, MLE = {
            dataSet.MCAR.imputed = impute.wrapper.MLE(dataSet.MCAR)
        }, SVD = {
            dataSet.MCAR.imputed = impute.wrapper.SVD(dataSet.MCAR,
                K = 2)
        }, KNN = {
            dataSet.MCAR.imputed = impute.wrapper.KNN(dataSet.MCAR,
                K = 15)
        })
        dataSet.imputed = dataSet.mvs
        dataSet.imputed[which(model.selector[[1]] == 1), ] = dataSet.MCAR.imputed
    }
    return(dataSet.imputed)
  }
```

`impute.MAR.MNAR` *Hybrid imputation method*

Description

Performs the imputation of missing values assuming that the missing values are both MAR and MNAR. It is assumed that the MNAR missing values are left-censored, more precisely, only rows (proteins/peptides) with a mean value below a censoring threshold are considered to contain left-censored missing data. The method relies on a estimation of the left-censoring threshold that is further used to distinguish rows (proteins/peptides) that contain left-censored missing data from those lines who contain random missing data.

Usage

```
impute.MAR.MNAR(dataSet.mvs, model.selector, method.MAR = "KNN", method.MNAR = "QRILC")
```

Arguments

`dataSet.mvs`	A data matrix containing missing values.
`model.selector`	Vector containing the flag indicators allowing the selection of the appropriate method for the imputation of missing data: the rows (peptides/proteins) corresponding to "1" in the model.selector vector are treated using a MAR specific method while the ones corresponding to "0" are treated using a MNAR specific method.
`method.MAR`	The method employed for the imputation of MAR/MCAR missing data. Possible values:
`method.MNAR`	The method employed for the imputation of the left-censored missing data.

Value

Data matrix containing complete abundances

Note

The method makes the assumption that rows (peptides/proteins) with a mean value lower than the censoring threshold are not affected my a random missingness mechanism. In reality, a random missingness mechanism could affect also rows (peptides/proteins) which are below the censoring threshold.

Author(s)

Cosmin Lazar

See Also

`model.Selector`, `impute.MAR`

Examples

```
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
```

```
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# run model.Selector
m.s = model.Selector(exprsData.MD)

# perform MAR-MNAR imputation
exprsData.MAR.imputed = impute.MAR.MNAR(exprsData.MD, m.s,
                           method.MAR = "KNN", method.MNAR = "QRILC")

## The function is currently defined as
function (dataSet.mvs, model.selector, method.MAR = "KNN", method.MNAR = "QRILC")
{
    switch(method.MAR, MLE = {
        dataSet.MCAR.imputed = impute.MAR(dataSet.mvs, model.selector,
            method = "MLE")
    }, SVD = {
        dataSet.MCAR.imputed = impute.MAR(dataSet.mvs, model.selector,
            method = "SVD")
    }, KNN = {
        dataSet.MCAR.imputed = impute.MAR(dataSet.mvs, model.selector,
            method = "KNN")
    })
    switch(method.MNAR, QRILC = {
        dataSet.complete.obj = impute.QRILC(dataSet.MCAR.imputed,
            tune.sigma = 0.3)
        dataSet.complete = dataSet.complete.obj[[1]]
    }, MinDet = {
        dataSet.complete = impute.MinDet(dataSet.MCAR.imputed)
    }, MinProb = {
        dataSet.complete = impute.MinProb(dataSet.MCAR.imputed)
    })
    return(dataSet.complete)
  }
```

`impute.MinDet` *Imputation of left-censored missing data using a deterministic minimal value approach.*

Description

Performs the imputation of left-censored missing data using a deterministic minimal value approach.Considering a peptide/protein expression data matrix with `n` columns corresponding to biological samples and `p` lines corresponding to peptides/proteins, for each sample (column), the

missing entries are replaced with a minimal value observed in that sample. The minimal value observed is estimated as being the q-th quantile (e.g. q = 0.01) of the observed values in that sample.

Usage

```
impute.MinDet(dataSet.mvs, q = 0.01)
```

Arguments

dataSet.mvs	A data matrix containing left-censored missing data.
q	A scalar used to determine a low expression value to be used for missing data imputation. 0 < q < 1, in this case q should be set to a low value. The default value is q = 0.01.

Value

A complete expression data matrix with missing values imputed.

Author(s)

Cosmin Lazar

See Also

impute.QRILC, impute.MinProb, impute.ZERO

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random

m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# perform missing data imputation

exprsData.imputed = impute.MinDet(exprsData.MD)

## Not run:
hist(exprsData[,1])
```

```
hist(exprsData.MD[,1])
hist(exprsData.imputed[,1])

## End(Not run)

## The function is currently defined as
function (dataSet.mvs, q = 0.01)
{
    nSamples = dim(dataSet.mvs)[2]
    dataSet.imputed = dataSet.mvs
    lowQuantile.samples = apply(dataSet.imputed, 2, quantile,
        prob = q, na.rm = T)
    for (i in 1:(nSamples)) {
        dataSet.imputed[which(is.na(dataSet.mvs[, i])), i] = lowQuantile.samples[i]
    }
    return(dataSet.imputed)
  }
```

`impute.MinProb` *Imputation of left-censored missing data using stochastic minimal value approach.*

Description

Performs the imputation of left-censored missing data by random draws from a Gaussian distribution centered in a minimal value. Considering a peptide/protein expression data matrix with `n` columns corresponding to biological samples and `p` lines corresponding to peptides/proteins, for each sample (column), the mean value of the Gaussian distribution is set to a minimal value observed in that sample. The minimal value observed is estimated as being the `q-th` quantile (e.g. `q = 0.01`) of the observed values in that sample. The standard deviation is estimated as the median of the peptide/protein-wise standard deviations. Note that when estimating the standard deviation of the Gaussian distribution, only the peptides/proteins which present more than `50%` recorded values are considered.

Usage

```
impute.MinProb(dataSet.mvs, q = 0.01, tune.sigma = 1)
```

Arguments

`dataSet.mvs`	A data matrix containing left-censored missing data.
`q`	A scalar used to determine a low expression value to be used for missing data imputation. `0 < q < 1`, in this case q should be set to a low value. The default value is `q = 0.01`.
`tune.sigma`	A scalar used to control the standard deviation of the Gaussian distribution used for random draws. If the sd is overestimated, than `0 < sigma.coef < 1`. The default value is `tune.sigma = 1`.

Value

A complete expression data matrix with missing values imputed.

Author(s)

Cosmin Lazar

See Also

impute.QRILC, impute.MinDet, impute.ZERO

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 50
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# perform missing data imputation
exprsData.imputed = impute.MinProb(exprsData.MD,0.01,1)

## Not run:
hist(exprsData[,1])
hist(exprsData.MD[,1])
hist(exprsData.imputed[,1])

## End(Not run)

## The function is currently defined as
function (dataSet.mvs, q = 0.01, tune.sigma = 1)
{
    nSamples = dim(dataSet.mvs)[2]
    nFeatures = dim(dataSet.mvs)[1]
    dataSet.imputed = dataSet.mvs
    min.samples = apply(dataSet.imputed, 2, quantile, prob = q,
        na.rm = T)
    count.NAs = apply(!is.na(dataSet.mvs), 1, sum)
    count.NAs = count.NAs/nSamples
    dataSet.filtered = dataSet.mvs[which(count.NAs > 0.5), ]
    protSD = apply(dataSet.filtered, 1, sd)
    sd.temp = median(protSD, na.rm = T) * tune.sigma
```

```
  print(sd.temp)
  for (i in 1:(nSamples)) {
      dataSet.to.impute.temp = rnorm(nFeatures,
                                     mean = min.samples[i],
                                     sd = sd.temp)
      dataSet.imputed[which(is.na(dataSet.mvs[, i])), i] =
      dataSet.to.impute.temp[which(is.na(dataSet.mvs[,i]))]
  }
  return(dataSet.imputed)
}
```

`impute.QRILC` *Imputation of left-censored missing data using QRILC method.*

Description

This function implements QRILC, a missing data imputation method that performs the imputation of left-censored missing data using random draws from a truncated distribution with parameters estimated using quantile regression.

Usage

```
impute.QRILC(dataSet.mvs, tune.sigma = 1)
```

Arguments

`dataSet.mvs`	A data matrix containing left-censored missing data.
`tune.sigma`	A scalar used to tune the standard deviation (if the complete data distribution is not Gaussian). The default value is `tune.sigma = 1`, and it corresponds to the case where the complete data distribution is Gaussian.

Value

A list including elements:

`dataSet.imputed`	A complete expression data matrix with missing values imputed.
`QR.obj`	A `lm` object that contains the results of the QQ regression, denoting the mean and the standard deviation of the complete data distribution.

Author(s)

Cosmin Lazar

References

QRILC: a quantile regression approach for the imputation of left-censored missing data in quantitative proteomics, Cosmin Lazar et al. - to be submitted.

See Also

impute.MinDet, impute.MinProb, impute.ZERO

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# perform missing data imputation
obj.QRILC = impute.QRILC(exprsData.MD)
exprsData.imputed = obj.QRILC[[1]]

## Not run:
hist(exprsData[,1])
hist(exprsData.MD[,1])
hist(exprsData.imputed[,1])

## End(Not run)

## The function is currently defined as
function (dataSet.mvs, tune.sigma = 1)
{
    nFeatures = dim(dataSet.mvs)[1]
    nSamples = dim(dataSet.mvs)[2]
    dataSet.imputed = dataSet.mvs
    QR.obj = list()
    for (i in 1:nSamples) {
        curr.sample = dataSet.mvs[, i]
        pNAs = length(which(is.na(curr.sample)))/length(curr.sample)
        upper.q = 0.95
        q.normal = qnorm(seq(pNAs, upper.q, (upper.q - pNAs)/(upper.q *
            10000)), mean = 0, sd = 1)
        q.curr.sample = quantile(curr.sample, probs = seq(0,
            upper.q, 1e-04), na.rm = T)
        temp.QR = lm(q.curr.sample ~ q.normal)
        QR.obj[[i]] = temp.QR
        mean.CDD = temp.QR$coefficients[1]
        sd.CDD = as.numeric(temp.QR$coefficients[2])
        data.to.imp = rtmvnorm(n = nFeatures, mean = mean.CDD,
```

```
            sigma = sd.CDD * tune.sigma, upper = qnorm(pNAs,
                mean = mean.CDD, sd = sd.CDD), algorithm = c("gibbs"))
        curr.sample.imputed = curr.sample
      curr.sample.imputed[which(is.na(curr.sample))] = data.to.imp[which(is.na(curr.sample))]
        dataSet.imputed[, i] = curr.sample.imputed
    }
    results = list(dataSet.imputed, QR.obj)
    return(results)
  }
```

`impute.wrapper.KNN` *Wrapper function for KNN imputation.*

Description

This function is a wrapper around the `impute.knn` function from impute package that performs KNN imputation for a data matrix containing missing entries.

Usage

```
impute.wrapper.KNN(dataSet.mvs, K)
```

Arguments

`dataSet.mvs`	A data matrix containing left-censored missing entries.
`K`	T he number of neighbors used to infer the missing data.

Details

Requires impute package.

Value

A complete expression data matrix with missing values imputed.

Author(s)

Cosmin Lazar

References

Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M., Brown, P. and Botstein, D., Imputing Missing Data for Gene Expression Arrays, Stanford University Statistics Department Technical report (1999), http://www-stat.stanford.edu/~hastie/Papers/missing.pdf

Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein and Russ B. Altman, Missing value estimation methods for DNA microarrays BIOINFORMATICS Vol. 17 no. 6, 2001 Pages 520-525

See Also

impute.knn, impute.wrapper.SVD, impute.wrapper.MLE

Examples

```
# generate expression data matrix

exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                           meanSamples = 0, sdSamples = 0.2,
                           nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                           meanDynRange = 20, sdDynRange = 1,
                           meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random

m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# perform missing data imputation

exprsData.imputed = impute.wrapper.KNN(exprsData.MD,15)

## The function is currently defined as
function (dataSet.mvs, K)
{
    resultKNN = impute.knn(dataSet.mvs, k = K, rowmax = 0.99,
        colmax = 0.99, maxp = 1500, rng.seed = sample(1:1000,
            1))
    dataSet.imputed = resultKNN[[1]]
    return(dataSet.imputed)
  }
```

---

impute.wrapper.MLE *MLE-based imputation of missing data.*

Description

Is is a wrapper function that performs the imputation of missing data using EM algorithm. The wrapper is built around the imp.norm function from the norm package.

Usage

```
impute.wrapper.MLE(dataSet.mvs)
```

Arguments

dataSet.mvs	A data matrix containing left-censored missing data.

Value

A complete expression data matrix with missing values imputed.

Author(s)

Cosmin Lazar

References

See package norm.

See Also

impute.wrapper.KNN, impute.wrapper.SVD

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random

m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# perform missing data imputation

exprsData.imputed = impute.wrapper.MLE(exprsData.MD)

## Not run:
hist(exprsData[,1])
hist(exprsData.MD[,1])
hist(exprsData.imputed[,1])

## End(Not run)

## The function is currently defined as
function (dataSet.mvs)
{
```

```
  s <- prelim.norm(dataSet.mvs)
  thetahat <- em.norm(s, showits = FALSE)
  rngseed(1234567)
  dataSet.imputed <- imp.norm(s, thetahat, dataSet.mvs)
  return(dataSet.imputed)
}
```

`impute.wrapper.SVD` *SVD-based imputation.*

Description

This is a wrapper function that performs SVD-based imputation of missing data. The wrapper is built around the pca function from the pcaMethods package.

Usage

```
impute.wrapper.SVD(dataSet.mvs, K)
```

Arguments

`dataSet.mvs`	A data matrix containing left-censored missing data.
`K`	The number of PCs used.

Value

A complete expression data matrix with missing values imputed.

Author(s)

Cosmin Lazar

References

See package pcaMethods

See Also

pca, impute.wrapper.KNN, impute.wrapper.MLE

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]
```

```
# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# perform missing data imputation
exprsData.imputed = impute.wrapper.SVD(exprsData.MD,2)

## Not run:
hist(exprsData[,1])
hist(exprsData.MD[,1])
hist(exprsData.imputed[,1])

## End(Not run)

## The function is currently defined as
function (dataSet.mvs, K)
{
    resultSVD = pca(dataSet.mvs, method = "svdImpute", nPcs = K)
    dataSet.imputed = resultSVD@completeObs
    return(dataSet.imputed)
  }
```

`impute.ZERO` *Imputation of missing entries by* `0`.

Description

This function performs the trivial imputation of missing values by `0`. Is is only used for comparison purposes.

Usage

```
impute.ZERO(dataSet.mvs)
```

Arguments

`dataSet.mvs`	A data matrix containing left-censored missing data.

Value

A complete expression data matrix with missing values imputed.

Author(s)

Cosmin Lazar

See Also

impute.QRILC, impute.MinDet, impute.MinProb

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# perform missing data imputation
exprsData.imputed = impute.ZERO(exprsData.MD)

## Not run:
hist(exprsData[,1])
hist(exprsData.MD[,1])
hist(exprsData.imputed[,1])

## End(Not run)

## The function is currently defined as
function (dataSet.mvs)
{
    dataSet.imputed = dataSet.mvs
    dataSet.imputed[which(is.na(dataSet.mvs))] = 0
    return(dataSet.imputed)
  }
```

`insertMVs` *Generates missing data in a complete data matrix.*

Description

This function generates missing data in a complete data matrix. Both random and left-censored missing data can be generated. The percentage of all missing data is controlled by `mean.THR`. The percentage of missing data which are left-censored is controlled by `MNAR.rate`.

Usage

```
insertMVs(original, mean.THR, sd.THR, MNAR.rate)
```

Arguments

original	Original complete data matrix of peptide/protein expression.
mean.THR	Mean value of the threshold distribution which controls the total missing data rate (mean.THR should be initially set such that the result of the initial thresholding, in terms of no. of NAs, equals the total missing values rate). Example: if one wants to generate 30% missing data, mean.THR can be set as follows: mean.THR = quantile(exprsData, probs = 0.3).
sd.THR	Standard deviation of the threshold distribution which controls the total missing data rate. sd.THR is usually set to a small value (e.g. 0.1).
MNAR.rate	Percentage of MVs which are missing not at random. Among the total number of missing data (NA_rate_TOTAL) generated by the initial threshold, a percentage of missing data that equals MNAR.rate is preserved as such; the remaining missing data are replaced with the original values. Next, a proportion of (NA_rate_TOTAL - MNAR.rate) missing data are generated randomly.

Value

A list including elements:

original	Original complete data matrix
original.mvs	Data matrix derived from the original by generating missing data
pNaNs	The percetage of missing data generated in the original complete dataset

Author(s)

Cosmin Lazar

See Also

generate.ExpressionData

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 50% missing not at random

m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 50
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]
```

```
## Not run:
hist(exprsData[,1])
hist(exprsData.MD[,1])
hist(exprsData.imputed[,1])

## End(Not run)

## The function is currently defined as
function (original, mean.THR, sd.THR, MNAR.rate)
{
    originalNaNs = original
    nProt = nrow(original)
    nSamples = ncol(original)
    thr = matrix(rnorm(nSamples * nProt, mean.THR, sd.THR), nProt,
        nSamples)
    indices.MNAR = which(original < thr)
    no.MNAR = round(MNAR.rate/100 * length(indices.MNAR))
    temp = matrix(original, 1, nSamples * nProt)
    temp[sample(indices.MNAR, no.MNAR)] = NaN
    indices.MCAR = which(!is.na(temp))
    no.MCAR = floor((100 - MNAR.rate)/100 * length(indices.MNAR))
    print(no.MCAR + no.MNAR)
    temp[sample(indices.MCAR, no.MCAR)] = NaN
    originalNaNs = matrix(temp, nProt, nSamples)
    originalNaNs_adjusted = originalNaNs
    noNaNs_Var = rowSums(is.na(originalNaNs))
    allNaNs_Vars = which(noNaNs_Var == nSamples)
    sampleIndexToReplace = sample(1:nSamples, length(allNaNs_Vars),
        replace = T)
    for (i in 0:length(sampleIndexToReplace)) {
      originalNaNs_adjusted[allNaNs_Vars[i], sampleIndexToReplace[i]] = original[allNaNs_Vars[i],
           sampleIndexToReplace[i]]
    }
    pNaNs = length(which(is.na(originalNaNs_adjusted)))/(nSamples *
        nProt)
    print(pNaNs)
    return(list(original, originalNaNs_adjusted, pNaNs))
  }
```

`intensity_PXD000022` *Dataset PXD000022 from ProteomeXchange.*

Description

This dataset has been collected during a study designed to compare the protein content of the exosome-like vesicles (ELVs) released from C2C12 murine myoblasts during proliferation (ELV-MB), and after differentiation into myotuves (ELV-MT). The dataset within this package contains proteins intensity processed using MaxQuant. More information can be found on ProteomeExchange public repository (http://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD000022) or in the original paper (see reference).

Usage

```
data(intensity_PXD000022)
```

Format

A data frame with 660 observations on the following 7 variables.

`Protein.IDs` Peptides/Proteins names

`Intensity.MB.1` a numeric vector

`Intensity.MB.2` a numeric vector

`Intensity.MB.3` a numeric vector

`Intensity.MT.1` a numeric vector

`Intensity.MT.2` a numeric vector

`Intensity.MT.3` a numeric vector

Source

Original MaxQuant data: http://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD000022

References

Forterre A, Jalabert A, Berger E, Baudet M, Chikh K, et al. (2014) Proteomic Analysis of C2C12 Myoblast and Myotube Exosome-Like Vesicles: A New Paradigm for Myoblast-Myotube Cross Talk? PLoS ONE 9(1): e84153. doi:10.1371/journal.pone.0084153

Examples

```
data(intensity_PXD000022)
```

---

intensity_PXD000052 *Dataset PXD000052 from ProteomeXchange.*

Description

This dataset has been collected during a study designed to perform the proteomic analysis of the SLP76 interactome in resting and activated primary mast cells. Four SLP76 replicates (with two analytical replicates each) have been affinity-purified from both resting and activated primary mast cells. The dataset within this package contains proteins intensity processed using MaxQuant. More information can be found on ProteomeExchange public repository (http://proteomecentral.proteomexchange.org/cgi/GetData
or in the original paper (see reference).

Usage

```
data(intensity_PXD000052)
```

Format

A data frame with 1991 observations on the following 17 variables.

`Protein.IDs` Peptides/Proteins names

`iBAQ.stSLP_activ1` a numeric vector

`iBAQ.stSLP_activ2` a numeric vector

`iBAQ.stSLP_activ3` a numeric vector

`iBAQ.stSLP_activ4` a numeric vector

`iBAQ.stSLP_rest1` a numeric vector

`iBAQ.stSLP_rest2` a numeric vector

`iBAQ.stSLP_rest3` a numeric vector

`iBAQ.stSLP_rest4` a numeric vector

`iBAQ.WT_activ1` a numeric vector

`iBAQ.WT_activ2` a numeric vector

`iBAQ.WT_activ3` a numeric vector

`iBAQ.WT_activ4` a numeric vector

`iBAQ.WT_rest1` a numeric vector

`iBAQ.WT_rest2` a numeric vector

`iBAQ.WT_rest3` a numeric vector

`iBAQ.WT_rest4` a numeric vector

Source

Original MaxQuant data: http://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD000052

References

Bounab Y, Hesse AM, Iannascoli B, Grieco L, Coute Y, Niarakis A, Roncagalli R, Lie E, Lam KP, Demangel C, Thieffry D, Garin J, Malissen B, Da?ron M, Proteomic analysis of the SH2 domain-containing leukocyte protein of 76 kDa (SLP76) interactome in resting and activated primary mast cells [corrected]. Mol Cell Proteomics, 12(10):2874-89(2013).

Examples

```
data(intensity_PXD000052)
```

`intensity_PXD000438` *Dataset PXD000438 from ProteomeXchange.*

Description

This dataset has been collected during a study designed to compare human primary tumor-derived xenograph proteomes of the two major histological non-small cel lung cancer subtypes: adenocarcinoma (ADC) and squamous cell carcinoma (SCC). The dataset within this package contains proteins intensity for 6 ADC and 6 SCC samples, processed using MaxQuant. More information can be found on ProteomeExchange public repository(http://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD00043 or in the original paper (see reference).

Usage

```
data(intensity_PXD000438)
```

Format

A data frame with 3709 observations on the following 13 variables.

`Protein.IDs` Peptides/Proteins names

`Intensity.092.1` a numeric vector

`Intensity.092.2` a numeric vector

`Intensity.092.3` a numeric vector

`Intensity.441.1` a numeric vector

`Intensity.441.2` a numeric vector

`Intensity.441.3` a numeric vector

`Intensity.561.1` a numeric vector

`Intensity.561.2` a numeric vector

`Intensity.561.3` a numeric vector

`Intensity.691.1` a numeric vector

`Intensity.691.2` a numeric vector

`Intensity.691.3` a numeric vector

Source

Original MaxQuant data: http://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD000438

References

Zhang W, Wei Y, Ignatchenko V, Li L, Sakashita S, Pham NA, Taylor P, Tsao MS, Kislinger T, Moran MF, Proteomic profiles of human lung adeno and squamous cell carcinoma using super-SILAC and label-free quantification approaches. Proteomics, 14(6):795-803(2014).

Examples

```
data(intensity_PXD000438)
```

`intensity_PXD000501` *Dataset PXD000501 from ProteomeXchange.*

Description

This dataset contains three biological replicates with three technical replicates each for the conditiones media (CM) and the whole cell lysates (WCL) of C8-D1A cell lines. The dataset within this package contains proteins iBAQ intensity processed using MaxQuant. More information can be found on ProteomeExchange public repository (http://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD00050 or in the original paper (see reference).

Usage

```
data(intensity_PXD000501)
```

Format

A data frame with 7363 observations on the following 19 variables.

`Protein.IDs` Peptides/Proteins names

`iBAQ.secretome_set1_tech1` a numeric vector

`iBAQ.secretome_set1_tech2` a numeric vector

`iBAQ.secretome_set1_tech3` a numeric vector

`iBAQ.secretome_set2_tech1` a numeric vector

`iBAQ.secretome_set2_tech2` a numeric vector

`iBAQ.secretome_set2_tech3` a numeric vector

`iBAQ.secretome_set3_tech1` a numeric vector

`iBAQ.secretome_set3_tech2` a numeric vector

`iBAQ.secretome_set3_tech3` a numeric vector

`iBAQ.whole_set1_tech1` a numeric vector

`iBAQ.whole_set1_tech2` a numeric vector

`iBAQ.whole_set1_tech3` a numeric vector

`iBAQ.whole_set2_tech1` a numeric vector

`iBAQ.whole_set2_tech2` a numeric vector

`iBAQ.whole_set2_tech3` a numeric vector

`iBAQ.whole_set3_tech1` a numeric vector

`iBAQ.whole_set3_tech2` a numeric vector

`iBAQ.whole_set3_tech3` a numeric vector

Source

Original MaxQuant data: http://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD000501

References

Han D, Jin J, Woo J, Min H, Kim Y, Proteomic analysis of mouse astrocytes and their secretome by a combination of FASP and StageTip-based, high pH, reversed-phase fractionation. Proteomics, ():(2014).

Examples

```
data(intensity_PXD000501)
```

---

model.Selector *Model selector for hybrid missing data imputation*

Description

The function sets a flag "1" for lines (proteins/peptides) where the imputation of missing values should be performed using a MAR/MCAR specific method, and "0" if the imputation should be performed using a MNAR specific method.

Usage

```
model.Selector(dataSet.mvs)
```

Arguments

dataSet.mvs	A data matrix containing missing values.

Value

A list including elements:

model.selector	Vector containing the flag indicators allowing the selection of the appropriate method for the imputation of missing data: the rows (peptides/proteins) corresponding to "1" in the model.selector vector are treated using a MAR specific method while the ones corresponding to "0" are treated using a MNAR specific method.
censoring.thr	The estimated censoring threshold.

Author(s)

Cosmin Lazar

See Also

impute.MAR, impute.MAR.MNAR

Examples

```
# generate expression data matrix
exprsDataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 1000, nFeaturesUp = 50, nFeaturesDown = 50,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = exprsDataObj[[1]]

# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# run model.Selector
m.s = model.Selector(exprsData.MD)

## The function is currently defined as
function (dataSet.mvs)
{
    nFeatures = dim(dataSet.mvs)[1]
    nSamples = dim(dataSet.mvs)[2]
    model.selector = rep(0, nFeatures)
    mean.vect = rowMeans(dataSet.mvs, na.rm = T)
    pNAs = length(which(is.na(mean.vect)))/length(mean.vect)
    upper.q = 0.99
    q.normal = qnorm(seq((pNAs + 0.001), (upper.q + 0.001), (upper.q -
        pNAs)/(upper.q * 100)), mean = 0, sd = 1)
    q.mean.vect = quantile(mean.vect, probs = seq(0.001, (upper.q +
        0.001), 0.01), na.rm = T)
    temp.QR = lm(q.mean.vect ~ q.normal)
    QR.obj = temp.QR
    mean.CDD = temp.QR$coefficients[1]
    sd.CDD = as.numeric(temp.QR$coefficients[2])
    nPoints = 512
    min.support = mean.CDD - 4 * sd.CDD
    max.support = mean.CDD + 4 * sd.CDD
    mean.vect.sorted = sort(mean.vect)
    Fn <- ecdf(mean.vect.sorted)
    support = c(seq(min.support, min(mean.vect, na.rm = T), length = nPoints),
        mean.vect.sorted, seq(max(mean.vect, na.rm = T), max.support,
            length = nPoints))
    cdf.OD = Fn(support)
    cdf.CD = pnorm(support, mean = mean.CDD, sd = sd.CDD)
    diff.cdf = (cdf.CD - cdf.OD)
    obj.fnc = (diff.cdf + 1)/(cdf.OD + 1) - 1
```

```
  obj.fnc = obj.fnc * 10
  if (max(obj.fnc) > 0) {
      censoring.thr = support[which(obj.fnc == max(obj.fnc))]
  }
  else {
      censoring.thr = min.support - 1
  }
  model.selector[which(mean.vect > censoring.thr)] = 1
  result = list(model.selector, censoring.thr)
  return(result)
}
```

`pep2prot` *Peptide to protein mapping.*

Description

This function performs peptide to protein mapping given a peptide expression matrix and a peptide to protein map.

Usage

```
pep2prot(pep.Expr.Data, rollup.map)
```

Arguments

`pep.Expr.Data`	Peptide expression data matrix.
`rollup.map`	Peptide to protein map.

Value

A protein expression matrix. Each line is obtained by aggregating the lines in the peptide expression matrix which have the same value in the rollup.map.

Author(s)

Cosmin Lazar

See Also

`pep2prot`, `generate.RollUpMap`

Examples

```
# generate expression data matrix
dataObj = generate.ExpressionData(nSamples1 = 6, nSamples2 = 6,
                          meanSamples = 0, sdSamples = 0.2,
                          nFeatures = 2000, nFeaturesUp = 100, nFeaturesDown = 100,
                          meanDynRange = 20, sdDynRange = 1,
                          meanDiffAbund = 1, sdDiffAbund = 0.2)
exprsData = dataObj[[1]]

# insert 15% missing data with 100% missing not at random
m.THR = quantile(exprsData, probs = 0.15)
sd.THR = 0.1
MNAR.rate = 100
exprsData.MD.obj = insertMVs(exprsData,m.THR,sd.THR,MNAR.rate)
exprsData.MD = exprsData.MD.obj[[2]]

# generate rollup.map
rollUpMap = generate.RollUpMap(round(dim(exprsData.MD)[1]/2),exprsData.MD)

# peptide to protein mapping
protExprsData = pep2prot(exprsData.MD,rollUpMap)

## The function is currently defined as
function (pep.Expr.Data, rollup.map)
{
    protIDs = unique(rollup.map)
    prot.Expr.Data = matrix(, nrow = length(protIDs), ncol = dim(pep.Expr.Data)[2])
    for (i in 1:length(protIDs)) {
        temp = protIDs[i]
        pepSet = pep.Expr.Data[which(rollup.map == temp), ]
        if (length(which(rollup.map == temp)) == 1) {
            prot.Expr.Data[i, ] = pepSet
        }
        else {
            prot.Expr.Data[i, ] = apply(pepSet, 2, median, na.rm = T)
        }
    }
    return(prot.Expr.Data)
  }
```

Index