Package 'matchFeat'

December 13, 2022

Type Package

Title One-to-One Feature Matching

Version 1.0

Date 2022-12-10

Author David Degras

Maintainer David Degras <david.degras@umb.edu>

Description Statistical methods to match feature vectors between multiple datasets in a one-to-one fashion. Given a fixed number of classes/distributions, for each unit, exactly one vector of each class is observed without label. The goal is to label the feature vectors using each label exactly once so to produce the best match across datasets, e.g. by minimizing the variability within classes. Statistical solutions based on empirical loss functions and probabilistic modeling are provided. The 'Gurobi' software and its 'R' interface package are required for one of the package functions (match.2x()) and can be obtained at <https://www.gurobi.com/> (free academic license). For more details, refer to Degras (2022) <doi:10.1080/10618600.2022.2074429> ``Scalable feature matching for large data collections'' and Bandelt, Maas, and Spieksma (2004) <doi:10.1057/palgrave.jors.2601723> ``Local search heuristics for multi-index assignment problems with decomposable costs''.

License GPL-2

Depends R (>= 3.5.0)

Imports clue, foreach, methods, utils

Suggests gurobi

NeedsCompilation no

Repository CRAN

Date/Publication 2022-12-13 12:30:07 UTC

R topics documented:

`matchFeat-package` *One-to-One Feature Matching*

Description

Statistical methods to match feature vectors between multiple datasets in a one-to-one fashion. Given a fixed number of classes/distributions, for each unit, exactly one vector of each class is observed without label. The goal is to label the feature vectors using each label exactly once so to produce the best match across datasets, e.g. by minimizing the variability within classes. Statistical solutions based on empirical loss functions and probabilistic modeling are provided. The 'Gurobi' software and its 'R' interface package are required for one of the package functions (match.2x()) and can be obtained at <https://www.gurobi.com/> (free academic license). For more details, refer to Degras (2022) <doi:10.1080/10618600.2022.2074429> "Scalable feature matching for large data collections" and Bandelt, Maas, and Spieksma (2004) <doi:10.1057/palgrave.jors.2601723> "Local search heuristics for multi-index assignment problems with decomposable costs".

Details

This package serves to match feature vectors across a collection of datasets in a one-to-one fashion. This task is formulated as a multidimensional assignment problem with decomposable costs (MDADC). We propose fast algorithms with time complexity roughly linear in the number n of datasets and space complexity a small fraction of the data size.

- Initialization methods: `match.rec` (recursive) and `match.template` (template-based).
- Main matching algorithms: `match.bca`, `match.bca.gen` (for unbalanced data), and `match.kmeans` (k-means matching).
- Refinement methods (post-processing): `match.2x` (pairwise interchange) and `match.gaussmix` (Gaussian mixture model with permutation constraints).

Author(s)

Author: David Degras
Maintainer: David Degras <david.degras@umb.edu>

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429
Wright (2015). Coordinate descent algorithms. `https://arxiv.org/abs/1502.04759`
McLachlan and Krishnan (2008). *The EM Algorithm and Extensions*

`match.2x` *Pairwise Interchange Heuristic (2-Assignment-Exchange)*

Description

This function implements the Pairwise Interchange Heuristic for the multidimensional assignment problem with decomposable costs (MDADC).

Usage

```
match.2x(x, sigma = NULL, unit = NULL, w = NULL, control = list())
```

Arguments

`x`	data: matrix of dimensions (mn, p) or 3D array of dimensions (p, m, n) with m = number of labels/classes, n = number of sample units, and p = number of variables)
`sigma`	permutations: matrix of dimensions (m, n)
`unit`	integer (=number of units) or vector mapping rows of `x` to sample units (length mn). Must be specified only if `x` is a matrix.
`w`	weights for loss function: single positive number, p-vector of length, or (p, p) positive definite matrix
`control`	tuning parameters

Details

Use of this function requires to have the GUROBI software and its R interface package installed. Both can be downloaded from `https://www.gurobi.com` after obtaining a free academic license.

Value

A list of class `matchFeat` with components

`sigma` best assignment as set of permutations ((m, n) matrix)

`cluster` best assignment as a cluster membership vector

`objective` minimum objective value

mu mean vector for each class/label ((p, m) matrix)

V covariance matrix for each class/label ((p, p, m) array)

call function call

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429

See Also

match.bca, match.bca.gen, match.gaussmix, match.kmeans, match.rec, match.template

Examples

```
if (require(gurobi)) {
## Generate small example
  m <- 3  # number of classes
  n <- 10 # number of statistical units
  p <- 5  # number of variables
  mu <- matrix(rnorm(p*m),p,m) # mean vectors
  sigma <- 0.1
  x <- array(as.vector(mu) + rnorm(p*m*n,sigma), c(p,m,n))

## Match all feature vectors
  result <- match.2x(x)

## Display results
  result$cost  # objective value = assignment cost
  result$sigma # solution permutations
  xmatched <- array(dim=dim(x))

## Matched feature vectors
  for (i in 1:n)
  xmatched[,,i] <- x[,result$sigma[,i],i]

}
```

match.bca *Block Coordinate Ascent Method*

Description

This function solves the multidimensional assignment problem with decomposable costs (MDADC) by block coordinate ascent. The dissimilarity function is the squared Euclidean distance.

Usage

```
match.bca(x, unit = NULL, w = NULL,
method = c("cyclical", "random"), control = list())
```

Arguments

`x` data: matrix of dimensions (mn, p) or 3D array of dimensions (p, m, n) with m = number of labels/classes, n = number of sample units, and p = number of variables)

`unit` integer (=number of units) or vector mapping rows of `x` to sample units (length mn). Must be specified only if `x` is a matrix.

`w` weights for loss function: single positive number, p-vector of length, or (p, p) positive definite matrix

`method` sweeping method for block coordinate ascent: `cyclical` or `random` (simple random sampling without replacement)

`control` tuning parameters

Details

Given a set of n statistical units, each having m possibly mislabeled feature vectors, the one-to-one matching problem is to find a set of n label permutations that produce the best match of feature vectors across units. The objective function to minimize is the sum of (weighted) squared Euclidean distances between all pairs of feature vectors having the same (new) label. This amounts to minimizing the sum of the within-label variances. The sample means and sample covariances of the matched feature vectors are calculated as a post-processing step.

The block-coordinate ascent (BCA) algorithm successively sweeps through the statistical units (=blocks), each time relabeling the m feature vectors of a unit to best match those of the other $n - 1$ units.

If `x` is a matrix, the rows should be sorted by increasing unit label and `unit` should be a nondecreasing sequence of integers, for example $(1, ..., 1, 2, ..., 2, ..., n, ..., n)$ with each integer $1, ..., n$ replicated m times.

The argument `w` can be specified as a vector of positive numbers (will be recycled to length p if needed) or as a positive definite matrix of size (p, p).

The optional argument `control` is a list with three fields: `sigma`, starting point for the optimization ((m, n) matrix of permutations; `maxit`, maximum number of iterations; and `equal.variance`, logical value that specifies whether the returned sample covariance matrices `V` for matched features should be equal between labels/classes (TRUE) or label-specific (FALSE, default).

Value

A list of class `matchFeat` with components

`sigma` best set of permutations for feature vectors ((m, n) matrix)

`cluster` associated clusters (=inverse permutations)

`objective` minimum objective value

`mu` sample mean for each class/label ((p, m) matrix)

`V` sample covariance for each class/label ((p, m) matrix

`call` function call

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429
Wright (2015). Coordinate descent algorithms. https://arxiv.org/abs/1502.04759

See Also

match.2x, match.bca.gen, match.gaussmix, match.kmeans, match.rec, match.template

Examples

```
data(optdigits)
m <- length(unique(optdigits$label)) # number of classes
n <- nrow(optdigits$x) / m # number of units

## Use function with data in matrix form
fit1 <- match.bca(optdigits$x, unit=n)

## Use function with data in array form
p <- ncol(optdigits$x)
x <- t(optdigits$x)
dim(x) <- c(p,m,n)
fit2 <- match.bca(x)
```

match.bca.gen *Block Coordinate Ascent Method for General (Balanced or Unbalanced) Data*

Description

Solve a feature matching problem by block coordinate ascent

Usage

```
match.bca.gen(x, unit = NULL, cluster = NULL, w = NULL,
method = c("cyclical", "random"), control = list())
```

Arguments

`x` data matrix (rows=instances, columns=features)

`unit` vector of unit labels (length = number of rows of `x`)

`cluster` integer specifying the number of classes/clusters to assign the feature vectors to OR integer vector specifying the initial cluster assignment.

`w` feature weights in loss function. Can be specified as single positive number, vector, or positive definite matrix

`method` sweeping method for block coordinate ascent: `cyclical` or `random` (simple random sampling without replacement)

`control` optional list of tuning parameters

Details

If `cluster` is an integer vector, it must have the same length as `unit` and its values must range between 1 and the number of clusters.

The list `control` can contain a field `maxit`, an integer that fixes the maximum number of algorithm iterations.

Value

A list of class `matchFeat` with components

`cluster` integer vector of cluster assignments (length = `now(x)`)

`objective` minimum objective value

`mu` sample mean for each cluster/class (feature-by-cluster matrix)

`V` sample covariance for each cluster/class (feature-by-feature-by-cluster 3D array)

`size` integer vector of cluster sizes

`call` function call

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429
Wright (2015). Coordinate descent algorithms. `https://arxiv.org/abs/1502.04759`

See Also

`match.2x`, `match.bca`, `match.bca.gen`, `match.gaussmix`, `match.kmeans`, `match.rec`, `match.template`

Examples

```
data(optdigits)
nobs <- nrow(optdigits$x) # total number of observations
n <- length(unique(optdigits$unit)) # number of statistical units
rmv <- sample.int(nobs, n-1) # remove (n-1) observations to make data unbalanced
min.m <- max(table(optdigits$unit[-rmv])) # smallest possible number of clusters
# lower values will result in an error message
m <- min.m
result <- match.bca.gen(optdigits$x[-rmv,], optdigits$unit[-rmv], m)
```

match.gaussmix *Gaussian Mixture Approach to One-To-One Feature Matching*

Description

This function performs maximum likelihood estimation (MLE) for the one-to-one feature matching problem represented as a multivariate Gaussian mixture model. MLE is carried out via the EM algorithm.

Usage

```
match.gaussmix(x, unit = NULL, mu = NULL, V = NULL, equal.variance = FALSE,
method = c("exact", "approx"), fixed = FALSE, control = list())
```

Arguments

`x`	data: matrix of dimensions (mn, p) or array of dimensions (p, m, n) with m = number of labels/classes, n = number of sample units, and p = number of variables)
`unit`	integer (=number of units) or vector mapping rows of `x` to sample units (length mn). Must be specified only if `x` is a matrix.
`mu`	matrix of initial estimates of mean vectors (dimension (p, m))
`V`	array of initial estimates of covariance matrices (dimension (p, p, m))
`equal.variance`	logical: if `TRUE`, all covariance matrices are assumed to be equal
`method`	method for calculating class probabilities of feature vectors
`fixed`	logical; if `TRUE`, the model parameters `mu` and `V` are fixed to their initial values
`control`	list of tuning parameters

Details

Given a sample of n statistical units, each having m possibly mislabeled feature vectors, the one-to-one matching problem is to find a set of n label permutations that produce the best match of feature vectors across units. This problem is sometimes referred to as "data association ambiguity".

The feature vectors of all units are represented as independent realizations of m multivariate normal distributions with unknown parameters. For each sample unit, exactly one vector from each distribution is observed and the m corresponding labels are randomly permuted. The goal is to estimate the true class of each feature vector, as well as the mean vector and covariance matrix of each distribution. These quantities are evaluated by ML estimation via the Expectation-Maximization (EM) algorithm.

If `x` is a matrix, the rows should be sorted by increasing unit label and `unit` should be a nondecreasing sequence of integers, for example $(1, ..., 1, 2, ..., 2, ..., n, ..., n)$ with each integer $1, ..., n$ replicated m times.

The arguments `mu` and `V` should be specified only if a good guess is available for these parameters. Otherwise bad starting values may cause the EM algorithm to converge to a local maximum of the likelihood function quite far from the global maximum.

If method is set to exact (default), the class probabilities of the feature vectors (given the data) are calculated exactly at each iteration of the EM algorithm. This operation can be slow as it involves calculating the permanent of matrices. The argument method can be set to approximate to speed up calculations, but this option is not recommended in general as the approximations used are very crude and may produce "bad" EM solutions.

The optional argument control can be specified with these fields: maxit, maximum number of EM iterations (default=1e4); eps, relative tolerance for EM convergence (default=1e-8), the EM algorithm stops if the relative increase in log-likelihood between two iterations is less than this tolerance; verbose, set to TRUE to display algorithm progress (default=FALSE).

Value

A list of class matchFeat with fields

sigma permutations that best match feature vectors across units ((m,n) matrix)

cluster associated clusters (=inverse permutations)

P conditional probability that a feature vector is assigned to its 'true' label ((m,n) matrix)

mu MLE of true mean vectors ((p,m) matrix)

V MLE of true covariance matrices ((p,p,m) array or (p,p)matrix if equal.variance=TRUE)

loglik Maximum value of log-likelihood

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429
‘McLachlan and Krishnan (2008). The EM Algorithm and Extensions.‘

See Also

match.2x, match.bca, match.bca.gen, match.kmeans, match.rec, match.template

Examples

```
data(optdigits)
x <- optdigits$x
label <- optdigits$label
m <- length(unique(label))
n <- length(unique(optdigits$unit))

## Randomly permute labels to make problem harder
for (i in 1:n)
{
idx <- seq.int((i-1) * m + 1, i * m)
sigma <- sample.int(m)
x[idx,] <- x[idx[sigma],]
label[idx] <- label[idx[sigma]]
}

## Fit Gaussian mixture model
```

```
fit <- match.gaussmix(x, unit = n)

## Calculate Rand index
Rand.index(fit$cluster,label)
```

match.kmeans *K-Means Matching Algorithm*

Description

This function matches collections of feature vectors in a one-to-one fashion using a k-means-like method.

Usage

```
match.kmeans(x, unit = NULL, w = NULL, method = c("hungarian", "bruteforce"),
control = list())
```

Arguments

x	data: matrix of dimensions (mn, p) or 3D array of dimensions (p, m, n) with m = number of labels/classes, n = number of sample units, and p = number of variables)
unit	integer (= number of units) or vector mapping rows of x to sample units (length mn). Must be specified only if x is a matrix.
w	weights for the (squared Euclidean) loss function. Can be specified as single positive number, p-vector, or $p \times p$ positive definite matrix
method	method for linear assignment problem: hungarian algorithm or bruteforce
control	optional list of tuning parameters

Details

Given a set of n units or datasets, each having m unlabeled feature vectors, the one-to-one matching problem is to find a set of n labels that produce the best match of feature vectors across units. The objective function to minimize is the sum of (weighted) squared Euclidean distances between all pairs of feature vectors having the same label. This amounts to minimizing the sum of the within-label variances. The sample means and sample covariances of the matched feature vectors are calculated as a post-processing step.

If x is a matrix, the rows should be sorted by increasing unit label and unit should be a nondecreasing sequence of integers, for example $(1, ..., 1, 2, ..., 2, ..., n, ..., n)$ with each integer $1, ..., n$ replicated m times.

The argument w can be specified as a vector of positive numbers (will be recycled to length p if needed) or as a positive definite matrix of size (p, p).

The optional argument control is a list with three fields: sigma, starting point for the optimization ((m, n) matrix of permutations; maxit, maximum number of iterations; and equal.variance, logical value that specifies whether the returned sample covariance matrices V for matched features should be equal between labels/classes (TRUE) or label-specific (FALSE, default).

Value

A list of class `matchFeat` with components

sigma best set of permutations for feature vectors ((m, n) matrix)

cluster associated clusters (= inverse permutations)

cost minimum objective value

mu sample mean for each class/label ((p, m) matrix)

V sample covariance for each class/label ((p, m) matrix

call function call

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429
`https://en.wikipedia.org/wiki/K-means_clustering`
`https://en.wikipedia.org/wiki/Assignment_problem`
`https://en.wikipedia.org/wiki/Hungarian_algorithm`

See Also

`match.2x`, `match.bca`, `match.bca.gen`, `match.gaussmix`, `match.rec`, `match.template`

Examples

```
## Generate data
  m <- 3
  n <- 10
  p <- 5
  mu <- matrix(rnorm(p*m),p,m)
  sigma <- 0.1
  x <- array(as.vector(mu) + rnorm(p*m*n,sigma), c(p,m,n))

## Match all feature vectors
  result <- match.kmeans(x)

## Display results
  result$objective # cost function
  xmatched <- array(dim=dim(x))

## re-arranged (matched) feature vectors
  for (i in 1:n){
  xmatched[,,i] <- x[,result$sigma[,i],i]}
```

match.rec *Recursive Initialization Method*

Description

RECUR1 algorithm of Bandelt et al (2004) to find starting point in the multidimensional assignment problem with decomposable costs (MDADC)

Usage

```
match.rec(x, unit = NULL, w = NULL, control = list())
```

Arguments

`x`	data: matrix of dimensions (mn, p) or 3D array of dimensions (p, m, n) with m = number of labels/classes, n = number of sample units, and p = number of variables)
`unit`	integer (=number of units) or vector mapping rows of `x` to sample units (length mn). Must be specified only if `x` is a matrix.
`w`	weights for loss function: single positive number, p-vector of length, or (p, p) positive definite matrix
`control`	tuning parameters

Value

A list of class `matchFeat` with components

`sigma` best set of permutations for feature vectors ((m, n) matrix)

`cluster` associated clusters (= inverse permutations)

`cost` minimum objective value

`mu` sample mean for each class/label ((p, m) matrix)

`V` sample covariance for each class/label ((p, m) matrix

`call` function call

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429
Bandelt, Maas, and Spieksma (2004), "Local search heuristics for multi-index assignment problems with decomposable costs." doi:10.1057/palgrave.jors.2601723

See Also

`match.2x`, `match.bca`, `match.gaussmix`, `match.template`, `match.kmeans`

Examples

```
data(optdigits)
m <- length(unique(optdigits$label)) # number of classes
n <- nrow(optdigits$x) / m # number of units

## Use function with data in matrix form
fit1 <- match.rec(optdigits$x, unit=n)

## Use function with data in array form
p <- ncol(optdigits$x)
x <- t(optdigits$x)
dim(x) <- c(p,m,n)
fit2 <- match.rec(x)
```

`match.template` *Template Matching*

Description

This function solves the multidimensional assignment problem with decomposable costs (MDADC) by matching the data to a pre-specified set of vectors (the template). The dissimilarity function is the squared Euclidean distance.

Usage

```
match.template(x, template = 1L, unit = NULL, w = NULL,
method = c("hungarian","bruteforce"), equal.variance = FALSE)
```

Arguments

`x` data: matrix of dimensions $mn \times p$ or 3D array of dimensions (p, m, n) with m = number of labels/classes, n = number of sample units, and p = number of variables)

`template` integer (= which sample unit to take as template) or (p, m) matrix

`unit` integer (=number of units/datasets) or vector mapping rows of `x` to sample units (length mn). Must be specified only if `x` is a matrix.

`w` weights for the loss function. Can be specified as a p-vector of diagonal weights or as a full $p \times p$ (positive definite) matrix

`method` method for the linear assignment problem: `hungarian` algorithm or `bruteforce`

`equal.variance` logical; if TRUE, resp. FALSE, return common, resp. label-specific, covariance of matched features

Details

Given n datasets or statistical units, each containing m feature vectors, the one-to-one matching problem is to find a set of n label permutations that produce the best match of feature vectors across units. The objective function to minimize is the sum of squared (Euclidean) distances between all feature vectors having the same (new) label. This amounts to minimizing the sum of the within-label variances.

The template-based method consists in relabeling successively each sample unit to best match a template matrix of feature vectors. This method is very fast but its optimization performance is only as good as the template. For best results, the template should be representative of the collected data.

If `x` is a matrix, the rows should be sorted by increasing unit label and `unit` should be a nondecreasing sequence of integers, for example $(1, ..., 1, 2, ..., 2, ..., n, ..., n)$ with each integer $1, ..., n$ replicated m times.

The argument `w` can be specified as a vector of positive numbers (will be recycled to length p if needed) or as a positive definite matrix of size (p, p).

Value

A list of class `matchFeat` with fields

`sigma` best assignement as set of permutations ($m \times n$ matrix)

`cluster` best assignement as cluster indicators ($m \times n$ matrix)

`objective` minimum objective value

`mu` mean vector for each class/label ($p \times m$ matrix)

`V` covariance matrix for each class/label ($p \times p \times m$ array if `equal.variance` is FALSE, $p \times p$ matrix otherwise

`call` function call

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429
https://en.wikipedia.org/wiki/Assignment_problem
https://en.wikipedia.org/wiki/Hungarian_algorithm

See Also

`match.2x`, `match.bca`, `match.bca.gen`, `match.gaussmix`, `match.kmeans`, `match.rec`

Examples

```
## Generate data
n <- 10
k <- 3
d <- 5
mu <- matrix(1:k, nrow=d, ncol=k, byrow=TRUE)
sigma <- 0.3
x <- array(mu, c(d,k,n)) + rnorm(d*k*n,sigma)
```

```
## Match all feature vectors with first case as template
result <- match.template(x,1)
## Display results
result$cost # cost function
xmatched <- array(dim=dim(x))
# re-arranged (matched) feature vectors
for (i in 1:n)
xmatched[,,i] <- x[,result$sigma[,i],i]
```

`objective.fun` *Calculate Cost of Multidimensional Assignment*

Description

Calculates the objective value in the multidimensional assignment problem with decomposable costs (MDADC). The dissimilarity function used in this problem is the squared Euclidean distance.

Usage

```
objective.fun(x, sigma = NULL, unit = NULL, w = NULL)
```

Arguments

`x` data: matrix of dimensions (mn, p) or 3D array of dimensions (p, m, n) with m = number of labels/classes, n = number of sample units, and p = number of variables)

`sigma` permutations: matrix of dimensions (m, n)

`unit` integer (=number of units) or vector mapping rows of `x` to sample units (length mn). Must be specified only if `x` is a matrix.

`w` weights for loss function: single positive number, p-vector of length, or (p, p) positive definite matrix

Details

Given n datasets having each m vectors of same size, say $x_{11}, ..., x_{1m}, ..., x_{n1}, ..., x_{nm}$, and permutations $\sigma_1, ..., \sigma_n$ of $1, ..., m$, the function calculates $1/(n(n{-}1))sum_{i,j}sum_k||x_{i,sigma_i(k)-x_{j,\sigma_j(k)}\|^2}$ where i and n run from 1 to n and k runs from 1 to m. This is the objective value (1) of Degras (2021), up to the factor $1/(n(n-1))$.

Value

Objective value

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429

See Also

objective.gen.fun

Examples

```
data(optdigits)
m <- 10
n <- 100
sigma <- matrix(1:m,m,n) # identity permutations
objective.fun(optdigits$x, sigma, optdigits$unit)
```

objective.gen.fun *Objective Value in One-To-One Feature Matching with Balanced or Unbalanced Data*

Description

Calculates the objective value in the multidimensional assignment problem with decomposable costs (MDADC). The dissimilarity function used in this problem is the squared Euclidean distance. The data can be balanced OR unbalanced.

Usage

```
objective.gen.fun(x, unit, cluster)
```

Arguments

x	data matrix with feature vectors in rows
unit	vector of unit labels (length should equal number of rows in x)
cluster	vector of cluster labels (length should equal number of rows in x)

Details

See equation (2) in Degras (2022). This function gives the same value as objective.fun when the data are balanced.

Value

Objective value

References

Degras (2022) "Scalable feature matching across large data collections." doi:10.1080/10618600.2022.2074429

See Also

objective.fun

Examples

```
data(optdigits)
m <- 10
n <- 100

## Balanced example: both 'objective.fun' and 'objective.gen.fun' work
sigma <- matrix(1:m,m,n)
cluster <- rep(1:m,n)
objective.fun(optdigits$x, sigma, optdigits$unit)
objective.gen.fun(optdigits$x, optdigits$unit, cluster)

## Unbalanced example
idx <- 1:999
objective.gen.fun(optdigits$x[idx,], optdigits$unit[idx], cluster[idx])
```

optdigits *Handwritten Digits Data*

Description

Digitized images of handwritten digits used in optical recognition tasks

Usage

```
data("optdigits")
```

Format

The format is:
List of 2
$ x : int [1:1000, 1:64] 0 0 0 0 0 0 0 0 0 0 ...
$ label: int [1:1000] 0 1 2 3 4 5 6 7 8 9 ...

Details

This is a subset of a larger dataset containing handwritten digits contributed by 30 people on a preprinted form. The forms were converted to normalized bitmaps of size 32x32 which were divided into nonoverlapping blocks of size 4x4. The number of pixels was counted in each block, producing a matrix of size 8x8 with integer coefficients ranging in 0..16. These matrix are vectorized in the rows of `optdigits$x`. The corresponding digits are in `optdigits$label`. 100 examples are available for each digit 0..9.

Source

UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

References

Alpaydin and Kaynak (1998). Cascading Classifiers. ftp://ftp.icsi.berkeley.edu/pub/ai/ethem/kyb.ps.Z

Examples

```
data(optdigits)
## Quick visualization
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(2,5))
for (i in 1:10) {
mat <- matrix(optdigits$x[i,],8,8)
image(mat[,8:1], xaxt="n", yaxt="n")
title(optdigits$label[i])
}
par(oldpar)
```

---

predict.matchFeat *Match New Feature Vectors To Existing Clusters*

Description

predict method for class "matchFeat"

Usage

```
## S3 method for class 'matchFeat'
predict(object, newdata, unit = NULL, ...)
```

Arguments

object	an object of class "matchFeat".
newdata	new dataset of feature vectors
unit	unit labels for new data. Only necessary if newdata is a matrix
...	for compatibility with the generic predict method; argument not currently used.

Details

The function predict.matchFeat finds the best matching for new feature vectors relative to an existing set of cluster/class centers. If codeobject results from a call to match.gaussmix, the same function is used for prediction (with fixed mean vectors and covariance matrices). In other cases, the function match.template is used for prediction.

Value

A list of class `matchFeat` with fields

`sigma` best matching as set of permutations ((m,n) matrix)

`cluster` best matching as cluster indicators ((m,n)-matrix)

`objective` minimum objective value

`mu` mean vector for each class/label ((p,m) matrix)

`V` covariance matrix for each class/label ((p,p,m) array if `equal.variance` is FALSE, (p,p) matrix otherwise

`call` function call

See Also

`print.matchFeat`, `summary.matchFeat`

Examples

```
data(optdigits)
train.result <- match.bca(optdigits$x[1:900,], optdigits$unit[1:900])
test.result <- predict(train.result, optdigits$x[901:1000,], optdigits$unit[901:1000])
test.result
```

`print.matchFeat` *Print a matchFeat Object*

Description

`print` method for class `"matchFeat"`.

Usage

```
## S3 method for class 'matchFeat'
print(x,...)
```

Arguments

`x`	an object of class `"matchFeat"`.
`...`	for compatibility with the generic `print` method; argument not currently used.

Details

The function `print.matchFeat` concisely displays the information of an object of class `"matchFeat"`. More precisely it shows the data range, bandwidth used in local polynomial estimation, and key information on SCB and statistical tests.

Value

No return value, called for side effects

See Also

predict.matchFeat, summary.matchFeat

Examples

```
data(optdigits)
result <- match.bca(optdigits$x, optdigits$unit)
print(result)
```

Rand.index *Rand Index of Agreement Between Two Partitions*

Description

Calculates the Rand Index between two partitions of a set

Usage

```
Rand.index(x, y)
```

Arguments

x	first partition vector
y	second partition vector

Details

The two vectors x and y must have equal length. Given a set S and two partitions X and Y of S, the Rand index is the proportion of pairs of elements in S (out of all pairs) that are either concordant in both X and Y (i.e., they belong to the same member of X and to the same member of Y) or discordant (i.e., not concordant) in both X and Y.

Value

The Rand index (not adjusted for chance)

References

W. M. Rand (1971). "Objective criteria for the evaluation of clustering methods"
https://en.wikipedia.org/wiki/Rand_index

Examples

```
## Example 1
x <- sample.int(3, 20, replace = TRUE)
y <- sample.int(3, 20, replace = TRUE)
table(x,y)
Rand.index(x,y)

## Example 2
data(optdigits)
label <- optdigits$label
m <- length(unique(label)) # 10
n <- length(unique(optdigits$unit)) # 100
dim(label) <- c(m,n)
p <- ncol(optdigits$x) # 64
x <- array(t(optdigits$x),c(p,m,n))
## Permute data and labels to make problem harder
for (i in 1:n) {
sigma <- sample.int(m)
x[,,i] <- x[,sigma,i]
label[,i] <- label[sigma,i]
}
## Compare Rand indices of matching methods
Rand.index(match.bca(x)$cluster, label)
Rand.index(match.rec(x)$cluster, label)
Rand.index(match.template(x)$cluster, label)
Rand.index(match.kmeans(x)$cluster, label)
```

`summary.matchFeat` *Summarize a matchFeat Object*

Description

`summary` method for class `"matchFeat"`

Usage

```
## S3 method for class 'matchFeat'
summary(object, ...)
```

Arguments

`object`	an object of class `"matchFeat"`
`...`	additional arguments; not currently used.

Details

The function summary.matchFeat displays all fields of a matchFeat object at the exception of x, y, par, nonpar, normscb, and bootscb which are potentially big. It provides information on the function call, data, local polynomial fit, SCB, and statistical tests.

Value

No return value, called for side effects

See Also

predict.matchFeat, print.matchFeat

Examples

```
data(optdigits)
result <- match.bca(optdigits$x, optdigits$unit)
summary(result)
```

Index