Package: IBclust (via r-universe)

June 9, 2026

Type Package

Title Information Bottleneck Methods for Clustering Mixed-Type Data

Version 1.4

Description Implements multiple variants of the Information Bottleneck ('IB') method for clustering datasets containing continuous, categorical (nominal/ordinal) and mixed-type variables. The package provides deterministic, agglomerative, generalised, sequential, and standard IB clustering algorithms that preserve relevant information while forming interpretable clusters. The Deterministic Information Bottleneck is described in Costa et al. (2026) <doi:10.1016/j.patcog.2026.113580>. The standard IB method originates from Tishby et al. (2000) <doi:10.48550/arXiv.physics/0004057>, the agglomerative variant from Slonim and Tishby (1999) <https://papers.nips.cc/paper/1651-agglomerative-information-bottleneck>, the generalised IB from Strouse and Schwab (2017) <doi:10.1162/NECO_a_00961>, and the sequential IB from Slonim et al. (2002) <doi:10.1145/564376.564401>.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports Rcpp, stats, utils, np, rje, Rdpack

Suggests mclust

LinkingTo Rcpp, RcppArmadillo, RcppEigen

RoxygenNote 7.3.3

RdMacros Rdpack

NeedsCompilation yes

Config/pak/sysreqs make

Repository https://amarkos.r-universe.dev

Date/Publication 2026-06-09 21:27:09 UTC

RemoteUrl https://github.com/amarkos/ibclust

RemoteRef HEAD

RemoteSha 2700ef9507beb9fa06e230b53d5d954e5f141554

Contents

`IBclust-package` *IBclust: Information Bottleneck Clustering for Mixed-Type Data*

Description

The **IBclust** package provides clustering methods based on the Information Bottleneck (IB) principle, supporting continuous, categorical (nominal and ordinal), and mixed-type data. Four algorithmic variants are implemented: `AIBmix` for hierarchical (agglomerative) clustering, `DIBmix` for hard partitional clustering, `IBmix` for soft (fuzzy) clustering, and `GIBmix` for the generalised case interpolating between the two.

Choosing a method

All four methods share the same kernel similarity machinery; they differ in the form of the partition produced and the objective optimised:

- `AIBmix`: hierarchical, hard clustering via greedy bottom-up merging that minimises Jensen-Shannon divergence between cluster distributions. Returns a full hierarchy from n singletons to one cluster.
- `DIBmix`: hard partitional clustering for a fixed number of clusters; uses dynamic regularisation to guarantee non-empty clusters.
- `IBmix`: soft (fuzzy) partitional clustering; each observation has a probability vector over clusters.

- `GIBmix`: generalised case with a tunable fuzziness parameter $\alpha \in [0, 1]$ that interpolates between `DIBmix` ($\alpha = 0$) and `IBmix` ($\alpha = 1$).

For mathematical details of each algorithm, see the references on the individual function help pages, or Costa et al. (2026) for a unified treatment.

Kernel functions

The functions in **IBclust** use kernel density estimation with generalised product kernels to construct the matrix $P_{Y|X}$ of kernel weights. Bandwidth parameters control the smoothness of the estimate and can be either user-supplied or automatically selected.

For continuous variables:

- *Gaussian (RBF) kernel (Silverman 1998):*

$$K_c\left(\frac{x - x'}{s}\right) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{(x - x')^2}{2s^2}\right\}, \quad s > 0.$$

- *Epanechnikov kernel (Epanechnikov 1969):*

$$K_c(x - x'; s) = \begin{cases} \frac{3}{4\sqrt{5}}\left(1 - \frac{(x-x')^2}{5s^2}\right), & \text{if } \frac{(x-x')^2}{s^2} < 5 \\ 0, & \text{otherwise} \end{cases}, \quad s > 0.$$

For nominal (unordered categorical) variables:

- *Aitchison & Aitken kernel (Aitchison and Aitken 1976):*

$$K_u(x = x'; \lambda) = \begin{cases} 1 - \lambda, & \text{if } x = x' \\ \frac{\lambda}{\ell - 1}, & \text{otherwise} \end{cases}, \quad 0 \leq \lambda \leq \frac{\ell - 1}{\ell}.$$

- *Li & Racine kernel (Ouyang et al. 2006):*

$$K_u(x = x'; \lambda) = \begin{cases} 1, & \text{if } x = x' \\ \lambda, & \text{otherwise} \end{cases}, \quad 0 \leq \lambda \leq 1.$$

For ordinal (ordered categorical) variables:

- *Li & Racine kernel (Li and Racine 2003):*

$$K_o(x = x'; \nu) = \begin{cases} 1, & \text{if } x = x' \\ \nu^{|x - x'|}, & \text{otherwise} \end{cases}, \quad 0 \leq \nu \leq 1.$$

- *Wang & van Ryzin kernel (Wang and Van Ryzin 1981):*

$$K_o(x = x'; \nu) = \begin{cases} 1 - \nu, & \text{if } x = x' \\ \frac{1-\nu}{2}\nu^{|x - x'|}, & \text{otherwise} \end{cases}, \quad 0 \leq \nu \leq 1.$$

ℓ is the number of levels of the categorical variable. For ordinal variables, the `lambda` argument is used to define ν. Bandwidths are automatically selected using the approach in Costa et al. (2026) when set to -1.

Nyström approximation

Constructing the kernel similarity matrix $P_{Y|X}$ has cost $O(n^2)$, which becomes prohibitive for large datasets. The partitional methods (`DIBmix`, `IBmix`, `GIBmix`) support a Nyström approximation (Williams and Seeger 2000) of the form $P_{Y|X} \approx CWC^\top$, where C is an $n \times m$ matrix of kernel weights between all observations and m landmark points, and W is the $m \times m$ kernel matrix over the landmarks alone. This reduces the cost of constructing $P_{Y|X}$ from $O(n^2)$ to $O(nm)$, with $m \ll n$. The heuristic $m \approx \sqrt{n}$ is used as the default. The approximation is enabled by setting `nystrom = TRUE` in the fit functions; the number of landmark points is controlled via `n_landmarks`, and specific landmark observations can be requested via `landmark_indices`. The Nyström approximation is not available for `AIBmix`, since its greedy merging procedure has cost $O(n^3)$ regardless of how $P_{Y|X}$ is constructed.

Author(s)

Maintainer: Angelos Markos <amarkos@gmail.com>

Authors:

- Efthymios Costa
- Ioanna Papatsouma

References

Aitchison J, Aitken CG (1976). "Multivariate binary discrimination by the kernel method." *Biometrika*, **63**(3), 413–420.

Li Q, Racine J (2003). "Nonparametric estimation of distributions with categorical and continuous data." *Journal of Multivariate Analysis*, **86**(2), 266–292.

Silverman BW (1998). *Density Estimation for Statistics and Data Analysis (1st Ed.)*. Routledge.

Ouyang D, Li Q, Racine J (2006). "Cross-validation and the estimation of probability distributions with categorical data." *Journal of Nonparametric Statistics*, **18**(1), 69–100.

Wang M, Van Ryzin J (1981). "A class of smooth estimators for discrete distributions." *Biometrika*, **68**(1), 301–309.

Epanechnikov VA (1969). "Non-parametric estimation of a multivariate probability density." *Theory of Probability & Its Applications*, **14**(1), 153–158.

Costa E, Papatsouma I, Markos A (2026). "A deterministic information bottleneck method for clustering mixed-type data." *Pattern Recognition*, **179**, 113580. ISSN 0031-3203. doi:10.1016/j.patcog.2026.113580.

Williams C, Seeger M (2000). "Using the Nyström method to speed up kernel machines." *Advances in Neural Information Processing Systems*, **13**.

aibclust-methods *Methods for aibclust objects*

Description

S3 methods available for "aibclust" objects, including extractors for the cluster assignments and model parameters, an information-metrics accessor, conversion methods to hclust and dendrogram, and diagnostic plotting.

Usage

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

## S3 method for class 'aibclust'
summary(object, k = NULL, m = NULL, ...)

## S3 method for class 'summary.aibclust'
print(x, ...)

## S3 method for class 'aibclust'
plot(
  x,
  type = c("dendrogram", "info", "importance", "similarity"),
  X = NULL,
  ncl = NULL,
  color_by_type = TRUE,
  col = NULL,
  colorbar = TRUE,
  main = NULL,
  labels = TRUE,
  ...
)

## S3 method for class 'aibclust'
fitted(object, ncl, ...)

## S3 method for class 'aibclust'
coef(object, ...)
```

Arguments

x	an aibclust object.
...	additional arguments passed to or from other methods.
object	an aibclust object.
k	Optional integer vector of cluster counts (cuts) to summarise (same as m).

m	Optional synonym for k; the number of clusters.
type	Plot type: "dendrogram" (default), "info" (information retained curve), "importance" (variable importance bar chart), or "similarity" (heatmap of the kernel similarity matrix $P_{Y\|X}$).
X	Original data frame used to fit x; required for type = "importance" and type = "similarity" when the fit was constructed with keep_data = FALSE. If the fit already contains the training data (keep_data = TRUE), any supplied X is ignored with a warning.
ncl	Number of clusters to use. Required for fitted (cut the hierarchy) and plot with type = "importance". For type = "similarity", used (if supplied) to reorder rows and columns of the similarity matrix by the partition at the chosen cut and to draw cluster-boundary boxes.
color_by_type	Logical; if TRUE, colour bars by variable type (continuous / nominal / ordinal). Defaults to TRUE. Used only by type = "importance".
col	Optional color (or, for type = "similarity", a colour palette vector).
colorbar	Logical; if TRUE (default), draw a horizontal colour scale below the similarity heatmap. Used only by type = "similarity".
main	Optional title.
labels	Logical; show labels on dendrogram.

Details

The following methods are available:

- print and summary: concise and detailed descriptions of the cluster hierarchy.
- fitted: extract the cluster partition at a requested number of clusters via the ncl argument.
- coef: extract the model's bandwidth hyperparameters (s, lambda).
- info_metrics: extract information-theoretic quantities. Optional ncl argument returns scalar values at the chosen cluster count.
- as.hclust and as.dendrogram: convert to standard hclust and dendrogram objects, enabling cutree() and dendrogram-based tools.
- plot: produce diagnostic plots (type = "dendrogram", "info", "importance", or "similarity").

See Also

AIBmix

AIBmix *Agglomerative Information Bottleneck Clustering for Mixed-Type Data*

Description

The AIBmix function implements the Agglomerative Information Bottleneck (AIB) algorithm for hierarchical clustering of datasets containing mixed-type variables, including categorical (nominal and ordinal) and continuous variables. This method merges clusters so that information retention is maximised at each step to create meaningful clusters, leveraging bandwidth parameters to handle different categorical data types (nominal and ordinal) effectively (Slonim and Tishby 1999).

Usage

```
AIBmix(
  X,
  s = -1,
  lambda = -1,
  scale = TRUE,
  contkernel = "gaussian",
  nomkernel = "aitchisonaitken",
  ordkernel = "liracine",
  cat_first = FALSE,
  keep_data = TRUE
)
```

Arguments

X	A data frame containing the data to be clustered. Variables should be of type numeric (for continuous variables), factor (for nominal variables) or ordered (for ordinal variables).
s	A numeric value or vector specifying the bandwidth parameter(s) for continuous variables. The values must be greater than 0. The default value is -1, which enables the automatic selection of optimal bandwidth(s). Argument is ignored when no variables are continuous.
lambda	A numeric value or vector specifying the bandwidth parameter for categorical variables. The default value is -1, which enables automatic determination of the optimal bandwidth. For nominal variables and nomkernel = 'aitchisonaitken', the maximum allowable value of lambda is $(l-1)/l$, where l represents the number of categories, whereas for nomkernel = 'liracine' the maximum allowable value is 1. For ordinal variables, the maximum allowable value of lambda is 1, regardless of what ordkernel is being used. Argument is ignored when all variables are continuous.
scale	A logical value indicating whether the continuous variables should be scaled to have unit variance before clustering. Defaults to TRUE. Argument is ignored when all variables are categorical.

contkernel	Kernel used for continuous variables. Can be one of gaussian (default) or epanechnikov. Argument is ignored when no variables are continuous.
nomkernel	Kernel used for nominal (unordered categorical) variables. Can be one of aitchisonaitken (default) or liracine. Argument is ignored when no variables are nominal.
ordkernel	Kernel used for ordinal (ordered categorical) variables. Can be one of liracine (default) or wangvanryzin. Argument is ignored when no variables are ordinal.
cat_first	A logical value indicating whether bandwidth selection is prioritised for the categorical variables, instead of the continuous. Defaults to FALSE. Set to TRUE if you suspect that the continuous variables are not informative of the cluster structure. Can only be TRUE when all bandwidths are selected automatically (i.e. s = -1, lambda = -1).
keep_data	Logical; if TRUE, the original input data X is stored in the returned object as training_data, enabling the use of predict() and certain plotting methods without re-passing the data. Defaults to FALSE to keep returned objects lightweight.

Details

The AIBmix function produces a hierarchical agglomerative clustering of the data while retaining maximal information about the original variable distributions. The Agglomerative Information Bottleneck algorithm uses an information-theoretic criterion to merge clusters so that information retention is maximised at each step, hence creating meaningful clusters with maximal information about the original distribution. Bandwidth parameters for categorical (nominal, ordinal) and continuous variables are adaptively determined if not provided. This process identifies stable and interpretable cluster assignments by maximizing mutual information while controlling complexity. The method is well-suited for datasets with mixed-type variables and integrates information from all variable types effectively.

See IBclust-package for details on the available kernel functions and their bandwidth parameters.

Value

An object of class "aibclust" representing the final clustering result. The returned object is a list with the following components:

merges	A data frame with 2 columns and n rows, showing which observations are merged at each step.
merge_costs	A numeric vector tracking the cost incurred by each merge $I(T_m;Y)-I(T_{m-1};Y)$.
partitions	A list containing n sub-lists. Each sub-list includes the cluster partition at each step.
I_T_Y	A numeric vector including the mutual information $I(T_m;Y)$ as the number of clusters m increases.
I_X_Y	A numeric value of the mutual information $I(X;Y)$ between observation indices and location.
info_ret	A numeric vector of length n including the fraction of the original information retained after each merge.
H_T	A numeric vector of length n; the entropy $H(T_m)$ at each cluster count m.

H_T_X	A numeric vector of length n; the conditional entropy $H(T_m\|X)$ at each cluster count. For agglomerative hard clustering this is zero throughout, since T_m is a deterministic function of X.
I_T_X	A numeric vector of length n; the mutual information $I(T_m; X)$ at each cluster count. Equals $H(T_m)$ for hard clustering.
s	A numeric vector of bandwidth parameters used for the continuous variables. A value of -1 is returned if all variables are categorical.
lambda	A numeric vector of bandwidth parameters used for the categorical variables. A value of -1 is returned if all variables are continuous.
call	The matched call.
n	Number of observations.
contcols	Indices of continuous columns in X.
catcols	Indices of categorical columns in X.
kernels	List with names of kernels used for continuous, nominal, and ordinal features.
obs_names	Names of rows in X; used for plotting the cluster hierarchy using a dendrogram.
scale	Logical indicating whether continuous variables were scaled to unit variance before clustering.
training_data	The original input data X, included only when keep_data = TRUE; NULL or absent otherwise.

An object of class "aibclust". See aibclust-methods for the available S3 methods (print, summary, plot, fitted, coef, info_metrics, as.hclust, as.dendrogram).

Author(s)

Efthymios Costa, Ioanna Papatsouma, Angelos Markos

References

Slonim N, Tishby N (1999). "Agglomerative Information Bottleneck." *Advances in Neural Information Processing Systems*, **12**.

Examples

```
# Example dataset with categorical, ordinal, and continuous variables
set.seed(123)
data_mix <- data.frame(
  cat_var = factor(sample(letters[1:3], 100, replace = TRUE)),    # Nominal categorical variable
  ord_var = factor(sample(c("low", "medium", "high"), 100, replace = TRUE),
                   levels = c("low", "medium", "high"),
                   ordered = TRUE),                                  # Ordinal variable
  cont_var1 = rnorm(100),                                          # Continuous variable 1
  cont_var2 = runif(100)                                           # Continuous variable 2
)

# Perform Mixed-Type Hierarchical Clustering with Agglomerative IB
result_mix <- AIBmix(X = data_mix, lambda = -1, s = -1, scale = TRUE)
```

```
# Print clustering results
plot(result_mix, type = "dendrogram", xlab = "", sub = "", cex = 0.5)  # Plot dendrogram
plot(result_mix, type = "info", col = "black", pch = 16)  # Plot info retention curve

# Simulated categorical data example
set.seed(123)
data_cat <- data.frame(
  Var1 = as.factor(sample(letters[1:3], 200, replace = TRUE)),  # Nominal variable
  Var2 = as.factor(sample(letters[4:6], 200, replace = TRUE)),  # Nominal variable
  Var3 = factor(sample(c("low", "medium", "high"), 200, replace = TRUE),
                levels = c("low", "medium", "high"), ordered = TRUE)  # Ordinal variable
)

# Run AIBmix with automatic lambda selection
result_cat <- AIBmix(X = data_cat, lambda = -1)
coef(result_cat)                       # Check bandwidths chosen
fitted(result_cat, ncl = 3)            # Partition at the 3-cluster cut
info_metrics(result_cat, ncl = 3)      # Information-theoretic quantities at 3 clusters
plot(result_cat, ncl = 3, type = "similarity") # Plot of similarity matrix at 3 clusters
plot(result_cat, type = "dendrogram", xlab = "", sub = "", cex = 0.5)  # Plot dendrogram

# Results summary
summary(result_cat)

# Simulated continuous data example
set.seed(123)
# Continuous data with 200 observations, 5 features
data_cont <- as.data.frame(matrix(rnorm(1000), ncol = 5))

# Run AIBmix with automatic bandwidth selection
result_cont <- AIBmix(X = data_cont, s = -1, scale = TRUE)

# Print concise summary of output
print(result_cont)

# Convert to hclust for standard tree tools
hc <- as.hclust(result_cont)
cutree(hc, k = 3)
```

`as.hclust.aibclust` *Convert an aibclust object to hclust or dendrogram*

Description

Enables use of standard hierarchical-clustering methods (e.g., `cutree`, `as.dendrogram`) on AIBmix output.

Usage

```
## S3 method for class 'aibclust'
as.hclust(x, ...)

## S3 method for class 'aibclust'
as.dendrogram(object, ...)
```

Arguments

x	An aibclust object.
...	Ignored.
object	An aibclust object.

Value

For as.hclust.aibclust, an object of class "hclust". For as.dendrogram.aibclust, an object of class "dendrogram".

DIBmix *Deterministic Information Bottleneck Clustering for Mixed-Type Data*

Description

The DIBmix function implements the Deterministic Information Bottleneck (DIB) algorithm for clustering datasets containing continuous, categorical (nominal and ordinal), and mixed-type variables. This method optimises an information-theoretic objective to preserve relevant information in the cluster assignments while achieving effective data compression (Costa et al. 2026).

Usage

```
DIBmix(
  X,
  ncl,
  randinit = NULL,
  s = -1,
  lambda = -1,
  scale = TRUE,
  maxiter = 100,
  nstart = 100,
  contkernel = "gaussian",
  nomkernel = "aitchisonaitken",
  ordkernel = "liracine",
  cat_first = FALSE,
  verbose = FALSE,
  nystrom = FALSE,
  n_landmarks = NULL,
```

```
  landmark_indices = NULL,
  keep_data = TRUE
)
```

Arguments

X	A data frame containing the input data to be clustered. It should include categorical variables (factor for nominal and ordered for ordinal) and continuous variables (numeric).
ncl	An integer specifying the number of clusters.
randinit	An optional vector specifying the initial cluster assignments. If NULL, cluster assignments are initialized randomly.
s	A numeric value or vector specifying the bandwidth parameter(s) for continuous variables. The values must be greater than 0. The default value is -1, which enables the automatic selection of optimal bandwidth(s). Argument is ignored when no variables are continuous.
lambda	A numeric value or vector specifying the bandwidth parameter for categorical variables. The default value is -1, which enables automatic determination of the optimal bandwidth. For nominal variables and nomkernel = 'aitchisonaitken', the maximum allowable value of lambda is $(l-1)/l$, where l represents the number of categories, whereas for nomkernel = 'liracine' the maximum allowable value is 1. For ordinal variables, the maximum allowable value of lambda is 1, regardless of what ordkernel is being used. Argument is ignored when all variables are continuous.
scale	A logical value indicating whether the continuous variables should be scaled to have unit variance before clustering. Defaults to TRUE. Argument is ignored when all variables are categorical.
maxiter	The maximum number of iterations allowed for the clustering algorithm. Defaults to 100.
nstart	The number of random initializations to run. The best clustering solution is returned. Defaults to 100.
contkernel	Kernel used for continuous variables. Can be one of gaussian (default) or epanechnikov. Argument is ignored when no variables are continuous.
nomkernel	Kernel used for nominal (unordered categorical) variables. Can be one of aitchisonaitken (default) or liracine. Argument is ignored when no variables are nominal.
ordkernel	Kernel used for ordinal (ordered categorical) variables. Can be one of liracine (default) or wangvanryzin. Argument is ignored when no variables are ordinal.
cat_first	A logical value indicating whether bandwidth selection is prioritised for the categorical variables, instead of the continuous. Defaults to FALSE. Set to TRUE if you suspect that the continuous variables are not informative of the cluster structure. Can only be TRUE when data is of mixed-type and all bandwidths are selected automatically (i.e. s = -1, lambda = -1).
verbose	Logical. Defaults to FALSE to suppress progress messages. Change to TRUE to print.

`nystrom` Logical. Indicates if the Nyström approximation for kernel Gram matrices is to be used for quicker implementation. Defaults to `FALSE`. Change to `TRUE` only for data sets with over 1000 observations.

`n_landmarks` Number of randomly drawn landmark points used for the Nyström approximation. Must be a positive integer less than the number of observations `nrow(X)`. Defaults to `NULL`, which selects $\lceil \sqrt{n} \rceil$ observations, where n is the number of observations. Argument is ignored if `nystrom = FALSE`.

`landmark_indices` Optional integer vector specifying the exact indices of observations to use as landmark points for the Nyström approximation. Must contain unique integers in $[1, n]$, where n is the number of observations. When provided, this overrides random landmark sampling; if `n_landmarks` is also supplied, its value must equal `length(landmark_indices)`. Defaults to `NULL`, in which case landmarks are sampled randomly. Argument is ignored if `nystrom = FALSE`.

`keep_data` Logical; if `TRUE`, the original input data `X` is stored in the returned object as `training_data`, enabling the use of `predict()` and certain plotting methods without re-passing the data. Defaults to `FALSE` to keep returned objects lightweight.

Details

The `DIBmix` function clusters data while retaining maximal information about the original variable distributions. The Deterministic Information Bottleneck algorithm optimises an information-theoretic objective that balances information preservation and compression. Bandwidth parameters for categorical (nominal, ordinal) and continuous variables are adaptively determined if not provided. This iterative process identifies stable and interpretable cluster assignments by maximising mutual information while controlling complexity. The method is well-suited for datasets with mixed-type variables and integrates information from all variable types effectively. For data sets with over a thousand observations ($n > 1000$), a Nyström approximation of the kernel Gram matrix can be enabled via `nystrom = TRUE`; see `IBclust-package` for details.

See `IBclust-package` for details on the available kernel functions and their bandwidth parameters.

Value

An object of class `"gibclust"`. See `gibclust-methods` for the available S3 methods (`print`, `summary`, `plot`, `fitted`, `coef`, `info_metrics`, `predict`).

Author(s)

Efthymios Costa, Ioanna Papatsouma, Angelos Markos

References

Costa E, Papatsouma I, Markos A (2026). "A deterministic information bottleneck method for clustering mixed-type data." *Pattern Recognition*, **179**, 113580. ISSN 0031-3203. doi:10.1016/j.patcog.2026.113580.

Examples

```
# Example 1: Basic Mixed-Type Clustering
set.seed(123)

# Create a more realistic dataset with mixed variable types
data_mix <- data.frame(
  # Categorical variables
  education = factor(sample(c("High School", "Bachelor", "Master", "PhD"), 150,
                            replace = TRUE, prob = c(0.4, 0.3, 0.2, 0.1))),
  employment = factor(sample(c("Full-time", "Part-time", "Unemployed"), 150,
                             replace = TRUE, prob = c(0.6, 0.25, 0.15))),

  # Ordinal variable
  satisfaction = factor(sample(c("Low", "Medium", "High"), 150, replace = TRUE),
                        levels = c("Low", "Medium", "High"), ordered = TRUE),

  # Continuous variables
  income = rlnorm(150, meanlog = 10, sdlog = 0.5),  # Log-normal income
  age = rnorm(150, mean = 35, sd = 10),             # Normally distributed age
  experience = rpois(150, lambda = 8)               # Years of experience
)

# Perform Mixed-Type Clustering
result_mix <- DIBmix(X = data_mix, ncl = 3, nstart = 5)

# View results
print(result_mix)
summary(result_mix)
table(fitted(result_mix))

# Example 2: Comparing cat_first parameter
# When categorical variables are more informative
result_cat_first <- DIBmix(X = data_mix, ncl = 3,
                           cat_first = TRUE,  # Prioritise categorical variables
                           nstart = 5)

# When continuous variables are more informative (default)
result_cont_first <- DIBmix(X = data_mix, ncl = 3,
                            cat_first = FALSE,
                            nstart = 5)

# Compare clustering performance
if (requireNamespace("mclust", quietly = TRUE)){  # For adjustedRandIndex
  print(paste("Agreement between approaches:",
              round(mclust::adjustedRandIndex(fitted(result_cat_first),
                    fitted(result_cont_first)), 3)))
  }

plot(result_cat_first, type = "sizes") # Bar plot of cluster sizes
plot(result_cat_first, type = "info")  # Information-theoretic quantities plot
plot(result_cat_first, type = "beta")  # Plot of evolution of beta
plot(result_cat_first, type = "importance") # Variable importance plot
```

```
plot(result_cat_first, type = "similarity") # Similarity matrix plot

# Simulated categorical data example
data_cat <- data.frame(
  Var1 = as.factor(sample(letters[1:3], 200, replace = TRUE)),  # Nominal variable
  Var2 = as.factor(sample(letters[4:6], 200, replace = TRUE)),  # Nominal variable
  Var3 = factor(sample(c("low", "medium", "high"), 200, replace = TRUE),
                levels = c("low", "medium", "high"), ordered = TRUE)  # Ordinal variable
)

# Perform hard clustering on categorical data with Deterministic IB
result_cat <- DIBmix(X = data_cat, ncl = 3, lambda = -1, nstart = 5)

# Print clustering results
fitted(result_cat)        # Cluster assignments
info_metrics(result_cat)  # Information-theoretic quantities
coef(result_cat)          # Hyperparameter values

# Simulated continuous data example
set.seed(123)
# Continuous data with 200 observations, 5 features
data_cont <- as.data.frame(matrix(rnorm(1000), ncol = 5))

# Perform hard clustering on continuous data with Deterministic IB
result_cont <- DIBmix(X = data_cont, ncl = 3, s = -1, nstart = 5)

# Print clustering results
fitted(result_cont)       # Cluster assignments

# Summary of output
print(result_cont)
summary(result_cont)
```

`find_elbow` *Detect a knee/elbow in a monotone curve*

Description

Identifies the point on a curve that lies at maximum perpendicular distance from the straight line connecting two endpoints. For information retention curves this is a common heuristic for choosing the number of clusters.

Usage

```
find_elbow(x, y, x_start, x_end)
```

Arguments

x	A numeric vector of x-coordinates, sorted in strictly ascending order.
y	A numeric vector of y-coordinates the same length as x, either non-decreasing or non-increasing throughout.
x_start	The x-coordinate of the segment's starting point. Must appear in x.
x_end	The x-coordinate of the segment's endpoint. Must appear in x, and satisfy x_end > x_start.

Details

The function expects x sorted in strictly ascending order, and y either non-decreasing or non-increasing throughout (no direction reversals). The endpoints x_start and x_end must be values present in x; the segment of the curve between them is searched for the elbow/knee.

Value

A list with components:

x	The x-coordinate of the detected elbow/knee.
y	The y-coordinate of the detected elbow/knee.
index	The index in the original x/y vectors.
distance	The perpendicular distance to the connecting line.

Examples

```
# Synthetic monotone-decreasing curve with a clear elbow
x <- 1:20
y <- 1 / (1 + exp(0.5 * (x - 5)))
elbow <- find_elbow(x, y, x_start = 1, x_end = 20)
elbow$x
plot(x, y, type = "b")
points(elbow$x, elbow$y, col = "red", pch = 19, cex = 1.5)

# Applied to an AIBmix information retention curve

fit <- AIBmix(iris[, -5])
find_elbow(seq_along(fit$info_ret), fit$info_ret,
           x_start = 1, x_end = 10)
```

gibclust-methods *Methods for gibclust objects*

Description

S3 methods available for "gibclust" objects, including extractors for the cluster assignments and model parameters, an information-metrics accessor, a prediction method for new data, and diagnostic plotting.

Usage

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

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

## S3 method for class 'summary.gibclust'
print(x, ...)

## S3 method for class 'gibclust'
plot(
  x,
  type = c("sizes", "info", "beta", "importance", "similarity", "membership"),
  X = NULL,
  color_by_type = TRUE,
  col = NULL,
  order_by_cluster = TRUE,
  groups = NULL,
  colorbar = TRUE,
  main = NULL,
  ...
)

## S3 method for class 'gibclust'
fitted(object, method = c("classes", "soft"), ...)

## S3 method for class 'gibclust'
coef(object, ...)
```

Arguments

x	a gibclust object.
...	additional arguments passed to or from other methods.
object	a gibclust object.

`type` Plot type: `"sizes"` (cluster sizes), `"info"` (information metrics), `"beta"` (log(beta) trajectory; DIBmix only), `"importance"` (variable importance bar chart), `"similarity"` (heatmap of the kernel similarity matrix $P_{Y|X}$), or `"membership"` (parallel coordinates plot illustrating cluster membership).

`X` Original data frame used to fit `x`; required for `type = "importance"` and `type = "similarity"` when the fit was constructed with `keep_data = FALSE`. If the fit already contains the training data (`keep_data = TRUE`), any supplied `X` is ignored with a warning.

`color_by_type` Logical; if `TRUE`, colour bars by variable type (continuous / nominal / ordinal). Defaults to `TRUE`. Used only by `type = "importance"`.

`col` Optional color (or, for type = "similarity" and type = "membership", a colour palette vector).

`order_by_cluster` Logical; if `TRUE` (default), rows and columns of the similarity matrix are reordered by cluster assignment and cluster-boundary boxes are drawn. Used only by `type = "similarity"`.

`groups` Optional grouping vector of length n used to colour the lines in `type = "membership"` (e.g. an external class label). If `NULL` (default), lines are coloured by the dominant (argmax) cluster. Used only by `type = "membership"`.

`colorbar` Logical; if `TRUE` (default), draw a horizontal colour scale below the similarity heatmap. Used only by `type = "similarity"`.

`main` Optional title.

`method` For `fitted`: either `"classes"` (default; hard cluster labels obtained via argmax of the membership matrix) or `"soft"` (the raw fuzzy membership matrix). For hard fits (DIBmix), `"classes"` returns the integer label vector and `"soft"` returns the equivalent one-hot binary matrix.

Details

The following methods are available:

- `print` and `summary`: concise and detailed descriptions of the cluster solution.
- `fitted`: extract cluster assignments. The argument `method = "classes"` (default) returns hard cluster labels; `method = "soft"` returns the membership matrix (one-hot for DIBmix; fuzzy for IBmix and GIBmix).
- `coef`: extract the model's hyperparameters as a list, including bandwidths (`s`, `lambda`) and IB parameters (`beta`, `alpha`).
- `info_metrics`: extract information-theoretic quantities $H(T)$, $H(T \mid X)$, $I(T;X)$, and $I(T;Y)$.
- `predict`: assign new observations to clusters. For DIBmix fits, returns hard labels via the minimisation of Kullback–Leibler divergence; for IBmix and GIBmix fits, returns a soft membership matrix via the soft IB assignment rule.
- `plot`: produce diagnostic plots (`type = "sizes"`, `"info"`, `"beta"`, `"importance"`, `"similarity"`, or `"membership"`).

See Also

DIBmix, IBmix, GIBmix

GIBmix *Generalised Information Bottleneck Clustering for Mixed-Type Data*

Description

The GIBmix function implements the Generalised Information Bottleneck (GIB) algorithm for clustering datasets containing continuous, categorical (nominal and ordinal), and mixed-type variables. This method optimises an information-theoretic objective to preserve relevant information in the cluster assignments while achieving effective data compression (Strouse and Schwab 2017).

Usage

```
GIBmix(
  X,
  ncl,
  beta,
  alpha,
  randinit = NULL,
  s = -1,
  lambda = -1,
  scale = TRUE,
  maxiter = 100,
  nstart = 100,
  conv_tol = 1e-05,
  contkernel = "gaussian",
  nomkernel = "aitchisonaitken",
  ordkernel = "liracine",
  cat_first = FALSE,
  verbose = FALSE,
  nystrom = FALSE,
  n_landmarks = NULL,
  landmark_indices = NULL,
  keep_data = TRUE
)
```

Arguments

X	A data frame containing the input data to be clustered. It should include categorical variables (factor for nominal and ordered for ordinal) and continuous variables (numeric).
ncl	An integer specifying the number of clusters.
beta	Regularisation strength characterising the tradeoff between compression and relevance. Must be non-negative.

`alpha` Strength of conditional entropy term. Must be in the range $[0, 1]$. Setting `alpha = 0` calls the `DIBmix` function and ignores the value of `beta`, while `alpha = 1` calls `IBmix` instead.

`randinit` An optional vector specifying the initial cluster assignments. If `NULL`, cluster assignments are initialised randomly.

`s` A numeric value or vector specifying the bandwidth parameter(s) for continuous variables. The values must be greater than 0. The default value is -1, which enables the automatic selection of optimal bandwidth(s). Argument is ignored when no variables are continuous.

`lambda` A numeric value or vector specifying the bandwidth parameter for categorical variables. The default value is -1, which enables automatic determination of the optimal bandwidth. For nominal variables and `nomkernel = 'aitchisonaitken'`, the maximum allowable value of `lambda` is $(l-1)/l$, where l represents the number of categories, whereas for `nomkernel = 'liracine'` the maximum allowable value is 1. For ordinal variables, the maximum allowable value of `lambda` is 1, regardless of what `ordkernel` is being used. Argument is ignored when all variables are continuous.

`scale` A logical value indicating whether the continuous variables should be scaled to have unit variance before clustering. Defaults to `TRUE`. Argument is ignored when all variables are categorical.

`maxiter` The maximum number of iterations allowed for the clustering algorithm. Defaults to 100.

`nstart` The number of random initialisations to run. The best clustering solution is returned. Defaults to 100.

`conv_tol` Convergence tolerance level; for a cluster membership matrix $U^{(m)}$ at iteration m, convergence is achieved if $\sum_{i,j}|U_{i,j}^{m+1} - U_{i,j}^{m}| \leq$ `conv_tol`. Must be in range $[0, 1]$. Defaults to `1e-5`.

`contkernel` Kernel used for continuous variables. Can be one of `gaussian` (default) or `epanechnikov`. Argument is ignored when no variables are continuous.

`nomkernel` Kernel used for nominal (unordered categorical) variables. Can be one of `aitchisonaitken` (default) or `liracine`. Argument is ignored when no variables are nominal.

`ordkernel` Kernel used for ordinal (ordered categorical) variables. Can be one of `liracine` (default) or `wangvanryzin`. Argument is ignored when no variables are ordinal.

`cat_first` A logical value indicating whether bandwidth selection is prioritised for the categorical variables, instead of the continuous. Defaults to `FALSE`. Set to `TRUE` if you suspect that the continuous variables are not informative of the cluster structure. Can only be `TRUE` when data is of mixed-type and all bandwidths are selected automatically (i.e. `s = -1`, `lambda = -1`).

`verbose` Logical. Defaults to `FALSE` to suppress progress messages. Change to `TRUE` to print.

`nystrom` Logical. Indicates if the Nyström approximation for kernel Gram matrices is to be used for quicker implementation. Defaults to `FALSE`. Change to `TRUE` only for data sets with over 1000 observations.

`n_landmarks` Number of randomly drawn landmark points used for the Nyström approximation. Must be a positive integer less than the number of observations `nrow(X)`. Defaults to `NULL`, which selects $\lceil \sqrt{n} \rceil$ observations, where n is the number of observations. Argument is ignored if `nystrom = FALSE`.

`landmark_indices` Optional integer vector specifying the exact indices of observations to use as landmark points for the Nyström approximation. Must contain unique integers in $[1, n]$, where n is the number of observations. When provided, this overrides random landmark sampling; if `n_landmarks` is also supplied, its value must equal `length(landmark_indices)`. Defaults to `NULL`, in which case landmarks are sampled randomly. Argument is ignored if `nystrom = FALSE`.

`keep_data` Logical; if `TRUE`, the original input data `X` is stored in the returned object as `training_data`, enabling the use of `predict()` and certain plotting methods without re-passing the data. Defaults to `FALSE` to keep returned objects lightweight.

Details

The `GIBmix` function produces a fuzzy clustering of the data while retaining maximal information about the original variable distributions. The Generalised Information Bottleneck algorithm optimises an information-theoretic objective that balances information preservation and compression. Bandwidth parameters for categorical (nominal, ordinal) and continuous variables are adaptively determined if not provided. This iterative process identifies stable and interpretable cluster assignments by maximising mutual information while controlling complexity. The method is well-suited for datasets with mixed-type variables and integrates information from all variable types effectively. Set $\alpha = 1$ and $\alpha = 0$ to recover the Information Bottleneck and its Deterministic variant, respectively. If $\alpha = 0$, the algorithm ignores the value of the regularisation parameter β. For data sets with over a thousand observations ($n > 1000$), a Nyström approximation of the kernel Gram matrix can be enabled via `nystrom = TRUE`; see `IBclust-package` for details.

See `IBclust-package` for details on the available kernel functions and their bandwidth parameters.

Value

An object of class `"gibclust"` representing the final clustering result. The returned object is a list with the following components:

`Cluster` An integer vector giving the cluster assignments for each data point.

`Entropy` A numeric value representing the entropy of the cluster assignments at convergence.

`CondEntropy` A numeric value representing the conditional entropy of cluster assignment, given the observation weights $H(T \mid X)$.

`MutualInfo` A numeric value representing the mutual information, $I(Y;T)$, between the original labels (Y) and the cluster assignments (T).

`InfoXT` A numeric value representing the mutual information, $I(X;T)$, between the original observations weights (X) and the cluster assignments (T).

`beta` A numeric vector of the final beta values used in the iterative procedure.

`alpha` A numeric value of the strength of conditional entropy used, controlling fuzziness of the solution.

s	A numeric vector of bandwidth parameters used for the continuous variables. A value of -1 is returned if all variables are categorical.
lambda	A numeric vector of bandwidth parameters used for the categorical variables. A value of -1 is returned if all variables are continuous.
call	The matched call.
ncl	Number of clusters.
n	Number of observations.
iters	Number of iterations used to obtain the returned solution.
converged	Logical indicating whether convergence was reached before maxiter.
conv_tol	Numeric convergence tolerance.
contcols	Indices of continuous columns in X.
catcols	Indices of categorical columns in X.
kernels	List with names of kernels used for continuous, nominal, and ordinal features.
nystrom_landmarks	Integer vector of observation indices used as landmark points when nystrom = TRUE; NULL otherwise.
scale	Logical indicating whether continuous variables were scaled to unit variance before clustering.
training_data	The original input data X, included only when keep_data = TRUE; NULL or absent otherwise.

An object of class "gibclust". See gibclust-methods for the available S3 methods (print, summary, plot, fitted, coef, info_metrics, predict).

Author(s)

Efthymios Costa, Ioanna Papatsouma, Angelos Markos

References

Strouse DJ, Schwab DJ (2017). "The Deterministic Information Bottleneck." *Neural Computation*, **29**(6), 1611–1630.

Examples

```
# Example dataset with categorical, ordinal, and continuous variables
set.seed(123)
data_mix <- data.frame(
 cat_var = factor(sample(letters[1:3], 100, replace = TRUE)),    # Nominal categorical variable
  ord_var = factor(sample(c("low", "medium", "high"), 100, replace = TRUE),
                   levels = c("low", "medium", "high"),
                   ordered = TRUE),                                  # Ordinal variable
  cont_var1 = rnorm(100),                                          # Continuous variable 1
  cont_var2 = runif(100)                                           # Continuous variable 2
)

# Perform Mixed-Type Fuzzy Clustering with Generalised IB
```

```
result_mix <- GIBmix(X = data_mix, ncl = 3, beta = 2, alpha = 0.5, nstart = 5)

# Print clustering results
fitted(result_mix, method = "soft")  # Cluster membership matrix
info_metrics(result_mix)             # Information-theoretic quantities
coef(result_mix)                     # Hyperperameters used

# Summary of output
summary(result_mix)

# Simulated categorical data example
set.seed(123)
data_cat <- data.frame(
  Var1 = as.factor(sample(letters[1:3], 200, replace = TRUE)),  # Nominal variable
  Var2 = as.factor(sample(letters[4:6], 200, replace = TRUE)),  # Nominal variable
  Var3 = factor(sample(c("low", "medium", "high"), 200, replace = TRUE),
                levels = c("low", "medium", "high"), ordered = TRUE)  # Ordinal variable
)

# Perform Fuzzy Clustering on categorical data with Generalised IB
result_cat <- GIBmix(X = data_cat, ncl = 2, beta = 25, alpha = 0.75, lambda = -1, nstart = 5)

# Print clustering results
fitted(result_cat, method = "soft")       # Cluster membership matrix
fitted(result_cat, method = "classes")    # Hardened cluster memberships

# Simulated continuous data example
set.seed(123)
# Continuous data with 200 observations, 5 features
data_cont <- as.data.frame(matrix(rnorm(1000), ncol = 5))

# Perform Fuzzy Clustering on continuous data with Generalised IB
result_cont <- GIBmix(X = data_cont, ncl = 2, beta = 50, alpha = 0.75, s = -1, nstart = 5)

# Print clustering results
print(result_cont)

plot(result_cont, type = "sizes") # Bar plot of cluster sizes (hardened assignments)
plot(result_cont, type = "info")  # Information-theoretic quantities plot
# Variable importance plot (hardened assignments)
plot(result_cont, type = "importance")
plot(result_cont, type = "membership") # Cluster membership plot
plot(result_cont, type = "similarity") # Similarity matrix plot
```

Description

The IBmix function implements the Information Bottleneck (IB) algorithm for clustering datasets containing continuous, categorical (nominal and ordinal), and mixed-type variables. This method optimises an information-theoretic objective to preserve relevant information in the cluster assignments while achieving effective data compression (Strouse and Schwab 2019).

Usage

```
IBmix(
  X,
  ncl,
  beta,
  randinit = NULL,
  s = -1,
  lambda = -1,
  scale = TRUE,
  maxiter = 100,
  nstart = 100,
  conv_tol = 1e-05,
  contkernel = "gaussian",
  nomkernel = "aitchisonaitken",
  ordkernel = "liracine",
  cat_first = FALSE,
  verbose = FALSE,
  nystrom = FALSE,
  n_landmarks = NULL,
  landmark_indices = NULL,
  keep_data = TRUE
)
```

Arguments

X	A data frame containing the input data to be clustered. It should include categorical variables (factor for nominal and ordered for ordinal) and continuous variables (numeric).
ncl	An integer specifying the number of clusters.
beta	Regularisation strength characterizing the tradeoff between compression and relevance. Must be non-negative.
randinit	An optional vector specifying the initial cluster assignments. If NULL, cluster assignments are initialized randomly.
s	A numeric value or vector specifying the bandwidth parameter(s) for continuous variables. The values must be greater than 0. The default value is -1, which enables the automatic selection of optimal bandwidth(s). Argument is ignored when no variables are continuous.
lambda	A numeric value or vector specifying the bandwidth parameter for categorical variables. The default value is -1, which enables automatic determination of the optimal bandwidth. For nominal variables and nomkernel = 'aitchisonaitken',

the maximum allowable value of `lambda` is $(l-1)/l$, where l represents the number of categories, whereas for `nomkernel = 'liracine'` the maximum allowable value is 1. For ordinal variables, the maximum allowable value of `lambda` is 1, regardless of what `ordkernel` is being used. Argument is ignored when all variables are continuous.

`scale` A logical value indicating whether the continuous variables should be scaled to have unit variance before clustering. Defaults to `TRUE`. Argument is ignored when all variables are categorical.

`maxiter` The maximum number of iterations allowed for the clustering algorithm. Defaults to 100.

`nstart` The number of random initialisations to run. The best clustering solution is returned. Defaults to 100.

`conv_tol` Convergence tolerance level; for a cluster membership matrix $U^{(m)}$ at iteration m, convergence is achieved if $\sum_{i,j}|U_{i,j}^{m+1} - U_{i,j}^{m}| \leq$ `conv_tol`. Must be in range $[0, 1]$. Defaults to `1e-5`.

`contkernel` Kernel used for continuous variables. Can be one of `gaussian` (default) or `epanechnikov`. Argument is ignored when no variables are continuous.

`nomkernel` Kernel used for nominal (unordered categorical) variables. Can be one of `aitchisonaitken` (default) or `liracine`. Argument is ignored when no variables are nominal.

`ordkernel` Kernel used for ordinal (ordered categorical) variables. Can be one of `liracine` (default) or `wangvanryzin`. Argument is ignored when no variables are ordinal.

`cat_first` A logical value indicating whether bandwidth selection is prioritised for the categorical variables, instead of the continuous. Defaults to `FALSE`. Set to `TRUE` if you suspect that the continuous variables are not informative of the cluster structure. Can only be `TRUE` when data is of mixed-type and all bandwidths are selected automatically (i.e. `s = -1`, `lambda = -1`).

`verbose` Logical. Defaults to `FALSE` to suppress progress messages. Change to `TRUE` to print.

`nystrom` Logical. Indicates if the Nyström approximation for kernel Gram matrices is to be used for quicker implementation. Defaults to `FALSE`. Change to `TRUE` only for data sets with over 1000 observations.

`n_landmarks` Number of randomly drawn landmark points used for the Nyström approximation. Must be a positive integer less than the number of observations `nrow(X)`. Defaults to `NULL`, which selects $\lceil\sqrt{n}\rceil$ observations, where n is the number of observations. Argument is ignored if `nystrom = FALSE`.

`landmark_indices`
Optional integer vector specifying the exact indices of observations to use as landmark points for the Nyström approximation. Must contain unique integers in $[1, n]$, where n is the number of observations. When provided, this overrides random landmark sampling; if `n_landmarks` is also supplied, its value must equal `length(landmark_indices)`. Defaults to `NULL`, in which case landmarks are sampled randomly. Argument is ignored if `nystrom = FALSE`.

`keep_data` Logical; if `TRUE`, the original input data `X` is stored in the returned object as `training_data`, enabling the use of `predict()` and certain plotting methods without re-passing the data. Defaults to `FALSE` to keep returned objects lightweight.

Details

The IBmix function produces a fuzzy clustering of the data while retaining maximal information about the original variable distributions. The Information Bottleneck algorithm optimises an information-theoretic objective that balances information preservation and compression. Bandwidth parameters for categorical (nominal, ordinal) and continuous variables are adaptively determined if not provided. This iterative process identifies stable and interpretable cluster assignments by maximising mutual information while controlling complexity. The method is well-suited for datasets with mixed-type variables and integrates information from all variable types effectively. For data sets with over a thousand observations ($n > 1000$), a Nyström approximation of the kernel Gram matrix can be enabled via nystrom = TRUE; see IBclust-package for details.

See IBclust-package for details on the available kernel functions and their bandwidth parameters.

Value

An object of class "gibclust" representing the final clustering result. The returned object is a list with the following components:

Cluster	An integer vector giving the cluster assignments for each data point.
Entropy	A numeric value representing the entropy of the cluster assignments at convergence.
CondEntropy	A numeric value representing the conditional entropy of cluster assignment, given the observation weights $H(T \mid X)$.
MutualInfo	A numeric value representing the mutual information, $I(Y;T)$, between the original labels (Y) and the cluster assignments (T).
InfoXT	A numeric value representing the mutual information, $I(X;T)$, between the original observations weights (X) and the cluster assignments (T).
beta	A numeric vector of the final beta values used in the iterative procedure.
alpha	A numeric value of the strength of conditional entropy used, controlling fuzziness of the solution. This is by default equal to 1 for IBmix.
s	A numeric vector of bandwidth parameters used for the continuous variables. A value of -1 is returned if all variables are categorical.
lambda	A numeric vector of bandwidth parameters used for the categorical variables. A value of -1 is returned if all variables are continuous.
call	The matched call.
ncl	Number of clusters.
n	Number of observations.
iters	Number of iterations used to obtain the returned solution.
converged	Logical indicating whether convergence was reached before maxiter.
conv_tol	Numeric convergence tolerance.
contcols	Indices of continuous columns in X.
catcols	Indices of categorical columns in X.
kernels	List with names of kernels used for continuous, nominal, and ordinal features.

`nystrom_landmarks`
: Integer vector of observation indices used as landmark points when `nystrom = TRUE`; `NULL` otherwise.

`scale`
: Logical indicating whether continuous variables were scaled to unit variance before clustering.

`training_data`
: The original input data `X`, included only when `keep_data = TRUE`; `NULL` or absent otherwise.

An object of class `"gibclust"`. See `gibclust-methods` for the available S3 methods (`print`, `summary`, `plot`, `fitted`, `coef`, `info_metrics`, `predict`).

Author(s)

Efthymios Costa, Ioanna Papatsouma, Angelos Markos

References

Strouse DJ, Schwab DJ (2019). "The information bottleneck and geometric clustering." *Neural Computation*, **31**(3), 596–612.

Examples

```
# Example dataset with categorical, ordinal, and continuous variables
set.seed(123)
data_mix <- data.frame(
 cat_var = factor(sample(letters[1:3], 100, replace = TRUE)),    # Nominal categorical variable
  ord_var = factor(sample(c("low", "medium", "high"), 100, replace = TRUE),
                   levels = c("low", "medium", "high"),
                   ordered = TRUE),                                  # Ordinal variable
  cont_var1 = rnorm(100),                                          # Continuous variable 1
  cont_var2 = runif(100)                                           # Continuous variable 2
)

# Perform Mixed-Type Fuzzy Clustering
result_mix <- IBmix(X = data_mix, ncl = 3, beta = 2, nstart = 1)

# Print clustering results
summary(result_mix)                    # Output summary
fitted(result_mix, method = "soft")    # Fuzzy clustering output
fitted(result_mix, method = "classes") # Hardened classes
coef(result_mix)                       # Hyperparameter values used
info_metrics(result_mix)               # Information-theoretic quantities


# Simulated categorical data example
set.seed(123)
data_cat <- data.frame(
  Var1 = as.factor(sample(letters[1:3], 100, replace = TRUE)),  # Nominal variable
  Var2 = as.factor(sample(letters[4:6], 100, replace = TRUE)),  # Nominal variable
  Var3 = factor(sample(c("low", "medium", "high"), 100, replace = TRUE),
                levels = c("low", "medium", "high"), ordered = TRUE)  # Ordinal variable
)
```

```
# Perform fuzzy clustering on categorical data with standard IB
result_cat <- IBmix(X = data_cat, ncl = 3, beta = 15, lambda = -1, nstart = 2, maxiter = 200)


plot(result_cat, type = "sizes") # Bar plot of cluster sizes (hardened assignments)
plot(result_cat, type = "info")  # Information-theoretic quantities plot
# Variable importance plot (hardened assignments)
plot(result_cat, type = "importance")
# Similarity plot
plot(result_cat, type = "similarity")
# Cluster membership plot
plot(result_cat, type = "membership")

# Simulated continuous data example
set.seed(123)
# Continuous data with 100 observations, 5 features
data_cont <- as.data.frame(matrix(rnorm(500), ncol = 5))

# Perform fuzzy clustering on continuous data with standard IB
result_cont <- IBmix(X = data_cont, ncl = 3, beta = 50, s = -1, nstart = 2)

# Print clustering results
print(result_cont)
```

`info_metrics` *Extract information-theoretic metrics from an IBclust fit*

Description

Returns the entropy, conditional entropy, and mutual information quantities computed by the chosen Information Bottleneck variant. Methods are provided for both `gibclust` and `aibclust` objects, returning a parallel set of quantities with class-specific extras.

Usage

```
info_metrics(object, ...)

## S3 method for class 'gibclust'
info_metrics(object, ...)

## S3 method for class 'aibclust'
info_metrics(object, ncl = NULL, ...)

## S3 method for class 'sibclust'
info_metrics(object, ...)
```

Arguments

object	A gibclust or aibclust object.
...	Additional arguments.
ncl	For aibclust objects, the number of clusters at which to evaluate the metrics. If NULL (default), returns the full vectors of metrics over all cluster counts. Ignored for gibclust objects.

Details

The shared quantities across both classes are:

- H_T: the entropy $H(T)$ of the cluster assignment.
- H_T_X: the conditional entropy $H(T \mid X)$ of the cluster assignment given the observation weights. This is 0 for hard partitions, since T is a deterministic function of X.
- I_T_X: the mutual information $I(T; X)$ between the cluster assignment and the observation weights. Equals H_T for hard partitions.
- I_T_Y: the mutual information $I(T; Y)$ between the cluster assignment and the joint distribution Y.

For gibclust objects, each quantity is a single numeric value corresponding to the fitted partition. For aibclust objects, each quantity is a numeric vector indexed by the number of clusters m, from $m = 1$ (a single cluster) to $m = n$ (all singletons). Supplying ncl extracts the scalar values at the chosen cut.

The aibclust output additionally includes:

- I_X_Y: the scalar baseline $I(X; Y)$, the mutual information between observation weights and the joint distribution.
- info_ret: the proportion of baseline information retained, $I(T_m; Y)/I(X; Y)$. Either a vector over m or a scalar at the requested cut.

Value

A named list of information-theoretic quantities. For gibclust, contains H_T, H_T_X, I_T_X, I_T_Y. For aibclust, contains those four quantities plus I_X_Y and info_ret.

See Also

DIBmix, IBmix, GIBmix, AIBmix.

Examples

```
# gibclust: single values per quantity
fit_dib <- DIBmix(iris[, -5], ncl = 3, nstart = 5)
info_metrics(fit_dib)

# aibclust: full vectors over cluster counts
fit_aib <- AIBmix(iris[, -5])
metrics_all <- info_metrics(fit_aib)
```

```
str(metrics_all)

# aibclust: scalar values at a specific cut
info_metrics(fit_aib, ncl = 3)
```

`predict.gibclust` *Predict cluster assignments for new observations*

Description

Assigns new observations to clusters using a fitted `gibclust` model. For hard fits (`DIBmix`, `alpha = 0`), returns integer cluster labels based on the minimisation of the Kullback–Leibler divergence between the new observation's conditional distribution and each cluster conditional density. For soft fits (`IBmix`, `GIBmix`), returns the full membership matrix based on the (G)IB update, with the fitted β.

Usage

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

Arguments

`object`	A fitted `gibclust` object produced by `DIBmix`, `IBmix`, or `GIBmix`.
`newdata`	A data frame of new observations to be assigned. Must have the same columns (with matching names and order) as the training data. If `NULL` (default), predictions are returned for the training data itself, providing a self-consistency check against `fitted(object)`.
`X`	The original training data frame used to fit `object`. Optional if `object` was constructed with `keep_data = TRUE`; in that case the stored training data is used automatically. Required otherwise.
`...`	Additional arguments (currently ignored).

Details

Prediction consists of three steps. First, the conditional distribution $p(y \mid x_{\text{new}})$ of each new observation over the training data is computed using the same kernel and bandwidths as the fit. Second, the cluster profiles $q(y \mid t)$ are reconstructed from the training data and the stored cluster assignments. Third, the Kullback–Leibler divergence $D_{KL}(p(y \mid x_{\text{new}}) \,||\, q(y \mid t))$ is computed for each (new observation, cluster) pair. New observations are then assigned via argmin (DIBmix) or via the soft assignment rule $q(t \mid x_{\text{new}}) \propto q(t) \exp(-\beta\, D_{KL})$ (IBmix and GIBmix).

Continuous variables in `newdata` are scaled using the means and standard deviations of the training data `X` when `object$scale` is `TRUE`. Categorical variables are relabelled to match the encoding used at fit time. New observations containing categorical levels not seen in the training data will produce an error.

Value

For DIBmix fits, an integer vector of length `nrow(newdata)` giving the predicted cluster label for each new observation. For IBmix and GIBmix fits, a numeric matrix of dimension `object$ncl` by `nrow(newdata)` containing soft membership probabilities, with columns summing to 1.

See Also

`DIBmix`, `IBmix`, `GIBmix`, `fitted`.

Examples

```
# Hold-out prediction on the iris data
set.seed(1)
test_idx  <- sample(seq_len(nrow(iris)), 30)
train_idx <- setdiff(seq_len(nrow(iris)), test_idx)

fit <- DIBmix(iris[train_idx, -5], ncl = 3, nstart = 5, keep_data = TRUE)
preds <- predict(fit, newdata = iris[test_idx, -5])
table(preds, iris$Species[test_idx])

# Grid prediction for visualising the decision boundary
fit2 <- DIBmix(faithful, ncl = 3, nstart = 5, keep_data = TRUE)
grid <- expand.grid(
  eruptions = seq(min(faithful$eruptions), max(faithful$eruptions), length = 50),
  waiting = seq(min(faithful$waiting), max(faithful$waiting), length = 50)
)
grid_pred <- predict(fit2, newdata = grid)
plot(grid, col = grid_pred + 1, pch = 15, cex = 0.6)
points(faithful, pch = fit2$Cluster, col = "black")
```

`predict.sibclust` *Predict cluster assignments for new observations*

Description

Assigns new observations to clusters using a fitted `sibclust` model. Each new observation is assigned to the cluster minimising the sequential merge cost, i.e. the (prior-weighted) loss of information incurred by absorbing the observation into the cluster, $(p_x + p_t)\,\mathrm{JS}_\pi(p(Y \mid x_{\text{new}}), q(Y \mid t))$, i.e. the same criterion optimised during fitting.

Usage

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

Arguments

object	A fitted sibclust object produced by sIBmix.
newdata	A data frame of new observations to be assigned. Must have the same columns (matching names and order) as the training data. If NULL (default), predictions are returned for the training data itself.
X	The original training data frame. Optional if object was constructed with keep_data = TRUE; required otherwise.
...	Additional arguments (currently ignored).

Details

Continuous variables in newdata are scaled using the means and standard deviations of the training data X when object$scale is TRUE. Categorical variables are relabelled to match the encoding used at fit time; new observations with categorical levels unseen in the training data produce an error.

Value

An integer vector of length nrow(newdata) giving the predicted cluster label for each new observation.

See Also

sIBmix, fitted.

---

sibclust-methods *Methods for sibclust objects*

Description

S3 methods available for "sibclust" objects, including extractors for the cluster assignments and model parameters, an information-metrics accessor, a prediction method for new data, and diagnostic plotting.

Usage

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

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

## S3 method for class 'summary.sibclust'
print(x, ...)

## S3 method for class 'sibclust'
plot(
```

```
  x,
  type = c("sizes", "info", "importance", "similarity"),
  X = NULL,
  color_by_type = TRUE,
  col = NULL,
  order_by_cluster = TRUE,
  colorbar = TRUE,
  main = NULL,
  ...
)

## S3 method for class 'sibclust'
fitted(object, method = c("classes", "soft"), ...)

## S3 method for class 'sibclust'
coef(object, ...)
```

Arguments

`x`	a `sibclust` object.
`...`	additional arguments passed to or from other methods.
`object`	a `sibclust` object.
`type`	Plot type: `"sizes"` (cluster sizes), `"info"` (information metrics), `"importance"` (variable importance bar chart), or `"similarity"` (heatmap of the kernel similarity matrix $P_{Y\|X}$).
`X`	Original data frame used to fit `x`; required for `type = "importance"` and `type = "similarity"` when the fit was constructed with `keep_data = FALSE`. If the fit already contains the training data (`keep_data = TRUE`), any supplied `X` is ignored with a warning.
`color_by_type`	Logical; if `TRUE`, colour bars by variable type (continuous / nominal / ordinal). Defaults to `TRUE`. Used only by `type = "importance"`.
`col`	Optional color (or, for `type = "similarity"`, a colour palette vector).
`order_by_cluster`	Logical; if `TRUE` (default), rows and columns of the similarity matrix are reordered by cluster assignment and cluster-boundary boxes are drawn. Used only by `type = "similarity"`.
`colorbar`	Logical; if `TRUE` (default), draw a horizontal colour scale below the similarity heatmap. Used only by `type = "similarity"`.
`main`	Optional title.
`method`	For `fitted`: either `"classes"` (default; the integer cluster-label vector) or `"soft"` (the equivalent one-hot binary membership matrix).

Details

The following methods are available:

- `print` and `summary`: concise and detailed descriptions of the cluster solution.

- `fitted`: extract cluster assignments. The argument `method = "classes"` (default) returns hard cluster labels; `method = "soft"` returns the equivalent one-hot membership matrix.
- `coef`: extract the model's bandwidth hyperparameters (`s`, `lambda`).
- `info_metrics`: extract information-theoretic quantities $H(T)$, $H(T \mid X)$, $I(T;X)$, and $I(T;Y)$.
- `predict`: assign new observations to clusters via the sequential merge-cost (Jensen–Shannon) rule.
- `plot`: produce diagnostic plots (`type = "sizes"`, `"info"`, `"importance"`, or `"similarity"`).

See Also

`sIBmix`

`sIBmix` *Sequential Information Bottleneck Clustering for Mixed-Type Data*

Description

The `sIBmix` function implements the sequential Information Bottleneck (sIB) algorithm for clustering datasets containing continuous, categorical (nominal and ordinal), and mixed-type variables. Unlike the agglomerative method (`AIBmix`), sIB maintains exactly `ncl` clusters throughout and is guaranteed to converge to a local maximum of the mutual information $I(Y;T)$ (Slonim et al. 2002).

Usage

```
sIBmix(
  X,
  ncl,
  randinit = NULL,
  s = -1,
  lambda = -1,
  scale = TRUE,
  maxiter = 100,
  nstart = 100,
  eps = 0,
  contkernel = "gaussian",
  nomkernel = "aitchisonaitken",
  ordkernel = "liracine",
  cat_first = FALSE,
  verbose = FALSE,
  keep_data = TRUE
)
```

Arguments

X	A data frame containing the input data to be clustered. It should include categorical variables (factor for nominal and ordered for ordinal) and continuous variables (numeric).
ncl	An integer specifying the number of clusters.
randinit	An optional vector specifying the initial cluster assignments. If NULL, cluster assignments are initialized randomly.
s	A numeric value or vector specifying the bandwidth parameter(s) for continuous variables. The values must be greater than 0. The default value is -1, which enables the automatic selection of optimal bandwidth(s). Argument is ignored when no variables are continuous.
lambda	A numeric value or vector specifying the bandwidth parameter for categorical variables. The default value is -1, which enables automatic determination of the optimal bandwidth. For nominal variables and nomkernel = 'aitchisonaitken', the maximum allowable value of lambda is $(l-1)/l$, where l represents the number of categories, whereas for nomkernel = 'liracine' the maximum allowable value is 1. For ordinal variables, the maximum allowable value of lambda is 1, regardless of what ordkernel is being used. Argument is ignored when all variables are continuous.
scale	A logical value indicating whether the continuous variables should be scaled to have unit variance before clustering. Defaults to TRUE. Argument is ignored when all variables are categorical.
maxiter	The maximum number of iterations allowed for the clustering algorithm. Defaults to 100.
nstart	The number of random initializations to run. The best clustering solution is returned. Defaults to 100.
eps	Convergence tolerance. A sweep producing at most eps * n re-assignments declares convergence. Defaults to 0 (a fully stable sweep), consistent with the hard-clustering convention of DIBmix.
contkernel	Kernel used for continuous variables. Can be one of gaussian (default) or epanechnikov. Argument is ignored when no variables are continuous.
nomkernel	Kernel used for nominal (unordered categorical) variables. Can be one of aitchisonaitken (default) or liracine. Argument is ignored when no variables are nominal.
ordkernel	Kernel used for ordinal (ordered categorical) variables. Can be one of liracine (default) or wangvanryzin. Argument is ignored when no variables are ordinal.
cat_first	A logical value indicating whether bandwidth selection is prioritised for the categorical variables, instead of the continuous. Defaults to FALSE. Set to TRUE if you suspect that the continuous variables are not informative of the cluster structure. Can only be TRUE when data is of mixed-type and all bandwidths are selected automatically (i.e. s = -1, lambda = -1).
verbose	Logical. Defaults to FALSE to suppress progress messages. Change to TRUE to print.
keep_data	Logical; if TRUE, the original input data X is stored in the returned object as training_data, enabling the use of predict() and certain plotting methods without re-passing the data. Defaults to FALSE to keep returned objects lightweight.

Details

The sIBmix function clusters data by maximising the information that the cluster assignments retain about the original variable distributions, while holding the number of clusters fixed at ncl. In contrast to the hierarchical agglomerative approach of AIBmix, the algorithm maintains a partition into exactly ncl clusters throughout. At each step a single observation is removed from its current cluster and re-assigned to the cluster for which the resulting loss of mutual information is smallest. Each such move can only increase the retained information, or leave it unchanged, so the procedure is guaranteed to converge to a local optimum. The criterion used to score each re-assignment is the same information-loss measure that underlies AIBmix, so the two methods optimise the same objective through different search strategies. Bandwidth parameters for categorical (nominal, ordinal) and continuous variables are adaptively determined if not provided. The algorithm is run from several random initial partitions (controlled by nstart) and the most informative solution is returned. The method is well-suited for datasets with mixed-type variables and integrates information from all variable types effectively.

See IBclust-package for details on the available kernel functions and their bandwidth parameters.

Value

An object of class "sibclust".

Author(s)

Efthymios Costa, Ioanna Papatsouma, Angelos Markos

References

Slonim N, Friedman N, Tishby N (2002). "Unsupervised document classification using sequential information maximization." In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 129–136.

Examples

```
set.seed(123)
data_mix <- data.frame(
  cat_var = factor(sample(letters[1:3], 100, replace = TRUE)),
  ord_var = factor(sample(c("low", "medium", "high"), 100, replace = TRUE),
                   levels = c("low", "medium", "high"), ordered = TRUE),
  cont_var1 = rnorm(100),
  cont_var2 = runif(100)
)

# Hard partitional clustering with sequential IB
result <- sIBmix(X = data_mix, ncl = 3, nstart = 5)

# Print output and provide summary
print(result)
summary(result)
fitted(result)              # Hard cluster labels
coef(result)                # Bandwidths
info_metrics(result)        # information-theoretic quantities
```

```
plot(result, type = "sizes")      # Plot of cluster sizes
plot(result, type = "info")       # Plot of information-theoretic quantities
plot(result, type = "importance") # Variable importance plot
plot(result, type = "similarity") # Similarity matrix plot

predict(result, newdata = data_mix[1:5, ])   # Predict integer labels for new data
```

Index