Package: FKmL (via r-universe)

May 11, 2026

Title Fréchet Distance-Based K-Means and Extensions for Longitudinal Data

Version 0.1.1

Description Implements shape-based clustering algorithms for multidimensional longitudinal data based on the Fréchet distance. It implements two main methods: MFKmL (Multidimensional Fréchet distance-based K-means for Longitudinal data), an extension of the K-means algorithm using the Fréchet distance originally developed in the 'kmlShape' package, adapted for multidimensional trajectories; and SFKmL (Sparse multidimensional Fréchet distance-based K-medoids for Longitudinal data), a K-medoids-based clustering algorithm that incorporates variable selection. These tools are designed to enhance clustering performance in high-dimensional longitudinal data settings, particularly those with time delays, variations in trajectory speed, irregular sampling intervals, and noise. This package implements methods derived from Kang et al. (2023) <doi:10.1007/s11222-023-10237-z>.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.2

Imports dplyr, ggplot2, proxy, abind

Depends R (>= 4.3)

NeedsCompilation no

Author Ji Hyun Park [aut, cre], Soon-Sun Kwon [aut], Ilsuk Kang [aut, ctb]

Maintainer Ji Hyun Park <jhn1105@gmail.com>

Repository https://jhp115.r-universe.dev

Date/Publication 2025-07-02 15:39:08 UTC

RemoteUrl https://github.com/cran/FKmL

RemoteRef HEAD

RemoteSha 6e58eca5fb7f7b5f2aeb947e48cb37e693212c64

Contents

`dist.array` *Compute Distance Array for Multidimensional Functional Data*

Description

This function standardizes multidimensional functional data using provided scaling factors, computes pairwise Fréchet distances between trajectories for each variable, and returns a distance array (3-dimensional array of distance matrices).

Usage

```
dist.array(dt, time_scale, var_scales)
```

Arguments

`dt`	A long-format data.frame containing the following columns in the specified order: • `ID`: An identifier for each trajectory. • `Time`: The time points at which measurements were recorded (numeric or integer vector). • `Variable1`, `Variable2`, ... : The measured variables over time (numeric values). The data.frame should not include any missing values. See 'Details' for structure requirements.
`time_scale`	A single numeric value used to scale the `Time` column. This ensures that time is appropriately weighted relative to the variables.
`var_scales`	A numeric vector of scaling factors for the measured variables. Its length must be equal to `ncol(dt) - 2`.

Details

The `dist.array` function first applies scaling to the `Time` and each measured variable. Then, it computes pairwise Fréchet distances between trajectories for each variable separately. The output is a 3-dimensional array in which each slice corresponds to a variable-specific distance matrix.

Unlike the `mfkml` function, which requires at least three measurements across time for each trajectory, the SFKmL ((Sparse multi-dimensional Fréchet distance-based K-medoids for Longitudinal data), which uses `dist.array`, allows for trajectories with missing values, as long as each variable has at least three time points for each trajectory. Therefore, `dt` may include missing values.

Value

A numeric value or matrix. If `form = "scalar"`, returns the Fréchet distance between the two trajectories as a single numeric value. If `form = "matrix"`, returns the dynamic programming matrix used to compute the distance.

A 3-dimensional array of pairwise distances with dimensions `[n, n, p]`, where:

n Number of unique trajectories.

p Number of variables.

Each slice `[, , k]` is a distance matrix for variable `k`.

`fredist` *Compute the Generalized Fréchet Distance Between Two Trajectories*

Description

Calculates the discrete Fréchet distance between two trajectories, which is used as the distance metric in clustering algorithms for longitudinal data.

Usage

```
fredist(traj1, traj2, form)
```

Arguments

`traj1`	A numeric matrix or data.frame representing the first trajectory. The first column must be time points, and the remaining columns should be one or more variables observed at each time point (e.g., Variable1, Variable2, ...). Each row corresponds to a single time point.
`traj2`	A numeric matrix or data.frame representing the second trajectory. The format should be the same as for `traj1`, where the first column is time, and the subsequent columns are variables.
`form`	A character string specifying the return format. Should be either `"scalar"` to return the scalar Fréchet distance, or `"matrix"` to return the full dynamic programming matrix.

Details

This function is primarily used internally by clustering functions to evaluate the similarity between trajectories based on the Fréchet distance. It is used in the `mfkml` function and for generating the distance array used in `SFclust` function.

Value

A numeric value or matrix. If `form = "scalar"`, returns the Fréchet distance between the two trajectories as a single numeric value. If `form = "matrix"`, returns the dynamic programming matrix used to compute the distance.

Examples

```
# Example trajectories with 3 variables
traj1 <- data.frame(
  Time = 1:4,
  Variable1 = c(1.2, 1.4, 1.6, 1.8),
  Variable2 = c(2.3, 2.1, 2.0, 1.9),
  Variable3 = c(3.1, 3.3, 3.5, 3.7)
)
traj2 <- data.frame(
  Time = 1:3,
  Variable1 = c(2.0, 2.2, 2.4),
  Variable2 = c(3.0, 2.9, 2.8),
  Variable3 = c(1.0, 1.1, 1.2)
)

# Compute Fréchet distance (scalar output)
fredist(traj1, traj2, form = "scalar")

# Compute Fréchet distance matrix
fredist(traj1, traj2, form = "matrix")
```

`mfkml` *Multidimensional Fréchet Distance-Based K-means for Longitudinal Data*

Description

Extends `kmlShape` to multidimensional ($p \geq 2$) longitudinal data. It performs scale adjustment and trajectory alignment across all variables prior to clustering to reduce distortions caused by differences in time grids and amplitude scales. When variables exhibit substantially different ranges, standardization is required to prevent any single variable from disproportionately influencing the clustering outcome.

The clustering process follows an iterative K-means framework, where cluster assignments are updated based on Fréchet distances. Cluster centers are computed using the weighted Fréchet mean, which accounts for variable weights assigned to individual trajectories. This allows the mean to be adjusted according to the relative importance of each trajectory in the clustering process.

Usage

```
mfkml(dt, clt_n, scales, weight, maxIter = 50)
```

Arguments

`dt` A long-format data.frame containing the following columns in the specified order:

- `ID`: An identifier for each trajectory.

- `Time`: The time points at which measurements were recorded (numeric or integer vector).
- `Variable1`, `Variable2`, ... : The measured variables over time (numeric values). The data.frame should not include any missing values. See 'Details' for structure requirements.

`clt_n` An integer specifying the number of clusters. The number of unique trajectories must be greater than or equal to `clt_n`.

`scales` A numeric vector used for scaling the time and variable columns. The length of `scales` must be equal to `ncol(dt) - 1`, where each value in `scales` corresponds to the scaling factor for the respective column (excluding the ID column). See 'Details' for structure requirements.

`weight` Specifies the weights used for calculating the weighted Fréchet mean. It can take one of the following forms:

- A data.frame with two columns: `ID` and `Weight`, where each `Weight` value indicates the importance of the corresponding trajectory.
- A numeric value of 1, indicating equal weights for all trajectories. See 'Details' for structure requirements.

`maxIter` The maximum number of iterations allowed before stopping if convergence is not reached. The default value is 50.

Details

The input dataset (`dt`) must contain only numeric values (except for the ID column) and must not include any missing values. Each variable should be measured at least three times per trajectory, since the method relies on trajectory shapes. Two observations per trajectory are insufficient to capture shape trends (e.g., increasing, decreasing, or stable).

Because the Fréchet distance is sensitive to measurement units, proper scaling is essential when applying the `mfkml` function. The `scales` vector contains scaling factors for time and each variable, which are used to rescale the corresponding columns. This scaling prevents distortion due to differences in the units of time and variables, allowing for more accurate shape-based comparisons.

This function involves random sampling internally. For reproducible results, set the random seed before calling the function using `set.seed()`.

Value

A list with the following components:

`Cluster` A data.frame containing the `ID` and `Cluster` columns, which indicate the final cluster assignment for each trajectory.

`Center` A data.frame representing the final cluster centers, with columns for the cluster IDs, time points, and variable values.

`Iteration` The number of iterations the algorithm performed before reaching convergence.

`SFclust` *Sparse Fréchet Distance-Based K-medoids for Longitudinal Data*

Description

Performs clustering on longitudinal trajectories using a sparse feature weighting scheme and Fréchet distance. The method iteratively updates cluster assignments and feature weights subject to an ℓ_1 norm constraint.

Usage

```
SFclust(k, l1bound, dist.ary, maxIter = 20, eps = 1e-04)
```

Arguments

`k`	The number of clusters.
`l1bound`	A bound on the ℓ_1 norm for the weight updates. It must lie between 1 and the square root of the number of variables.
`dist.ary`	A 3-dimensional array of pairwise Fréchet distances. The array should be of shape (n, n, p), where `n` is the number of trajectories and `p` is the number of variables. Each `dist.ary[,,j]` stores the pairwise distances for the `j`-th variable.
`maxIter`	The maximum number of iterations before stopping if convergence is not reached. Default is 20.
`eps`	A small positive threshold for convergence. The algorithm stops when the change in weights becomes smaller than this threshold. Default is 1e-4.

Details

The function assumes that the input `dist.array` contains pairwise distances between trajectories for each variable, using the generalized Fréchet distance. Clustering is performed via a k-medoids algorithm, and feature weights are updated using between-cluster sum of squares (BCSS) with sparsity control. If the number of variables is one, only clustering is performed, and no variable weighting is applied. This function involves random sampling internally. For reproducible results, set the random seed before calling the function using `set.seed()`.

Value

A list containing the following components:

clust A vector of cluster assignments for each trajectory.

final.weight The final weight vector after the last iteration, reflecting the contribution of each variable to the clustering process.

weight.history A matrix of weight values at each iteration, showing how the feature weights evolved.

criteria A vector of convergence criteria values for each iteration, quantifying the change in weights.

iteration The number of iterations performed before convergence or reaching `maxIter`.

`SFclust.permute` *Perform Permutation-Based Clustering Evaluation for SFclust*

Description

Performs a permutation-based analysis to evaluate clustering results across different values of the ℓ_1 norm constraint (`s`). This function is designed to help determine the most appropriate ℓ_1 norm value by comparing the observed clustering outcome with those obtained under random permutations.

The function computes gap statistics for each ℓ_1 norm constraint value based on permuted versions of the input distance array, and identifies the optimal `s` as the one maximizing the gap statistic. Two ggplot objects are returned to visualize the gap patterns.

Usage

```
SFclust.permute(dist.ary, k, nperms, l1b)
```

Arguments

`dist.ary`	A 3-dimensional distance array representing pairwise distances between trajectories across multiple variables. Follows the same format used in `SFclust`.
`k`	An integer specifying the number of clusters.
`nperms`	An integer specifying the number of permutations to perform.
`l1b`	A numeric vector of ℓ_1 norm constraint values to test during clustering. These values control the sparsity of the weights during clustering.

Details

This function helps assess the robustness of clustering structure and select an optimal level of sparsity. If any clustering attempt fails (e.g., due to convergence issues or weight update errors), the corresponding `l1b` values are reported in `failed_l1b` and `failed_j`. This function returns two ggplot objects (`gapplot.l1b` and `gapplot.nnz`) that can be used to visualize the gap statistics. These are not automatically printed, allowing users to decide when and how to display them. This function involves random sampling internally. For reproducible results, set the random seed before calling the function using `set.seed()`.

Value

A list containing the following components:

totss A numeric vector of total within-cluster sum of squared distances for each ℓ_1 norm value.

permtotss A matrix of total sum of squared distances for each permutation and each ℓ_1 norm value.

nnonzerowss A numeric vector of the number of nonzero weights for each ℓ_1 norm value.

gaps A numeric vector of gap statistics: the difference between observed and permuted clustering results.

sdgaps A numeric vector of standard deviations of the gaps across permutations.

l1bounds A vector of ℓ_1 norm constraint values that were successfully processed without error.

bestl1b The ℓ_1 norm constraint value that yielded the largest gap.

failed_j Indices of `l1b` values that caused errors during the clustering process.

failed_l1b The actual ℓ_1 norm values that caused errors.

gapplot.l1b A ggplot object showing the gap statistics plotted against ℓ_1 norm constraint values.

gapplot.nnz A ggplot object showing the gap statistics plotted against the number of nonzero weights.

Index