

Package ‘BIDistances’

August 21, 2026

Type Package

Title Bioinformatic Distances

Version 1.0.1

Date 2026-08-19

Maintainer Michael Thrun <m.thrun@gmx.net>

Description Provides a unified interface for computing, comparing, and examining distances, dissimilarities, divergences, and selected similarities for bioinformatics data. The core installation exposes 60 canonical named routes for numerical data and more than 90 when the suggested 'philentropy' backend is installed; aliases, user-defined functions, and mixed-data combinations are not included in these counts. Weighted Minkowski distances can be computed through 'parallelDist', an internal multicore implementation, or optional 'OpenCL' kernels, while the established weighted Euclidean GPU implementation is retained as the optimized $p = 2$ route. The package also supports theory-guided comparison of distance distributions for clustering, explicit mathematical property classifications, mixed-data constructions through 'manydist', and a specialized Gene Ontology-derived TF-IDF distance.

Depends R (>= 3.5.0)

Imports Rcpp (>= 1.0.8), RcppParallel (>= 5.1.1), parallelDist, parallel, DataVisualizations, diptest, e1071, pracma, ggplot2

Suggests knitr, rmarkdown, OpenCL, transport, ineq, ScatterDensity, memshare, philentropy (>= 0.10.0), dtw, proxy, manydist (>= 0.5.0), testthat (>= 3.0.0)

LinkingTo Rcpp, RcppParallel (>= 5.1.1)

NeedsCompilation yes

SystemRequirements GNU make, OpenCL library (optional, for GPU acceleration), pandoc (>=1.12.3, needed for vignettes)

License GPL-3

LazyLoad yes

LazyData TRUE

Encoding UTF-8

VignetteBuilder knitr

Config/testthat/edition 3

BugReports <https://github.com/Mthrun/BIDistances/issues>

Author Quirin Stier [aut, rev, ctb] (ORCID:

<<https://orcid.org/0000-0002-7896-4737>>),

Michael Thrun [aut, cre] (ORCID:

<<https://orcid.org/0000-0001-9542-5543>>),

Luca Brinkmann [ctb]

Repository CRAN

Date/Publication 2026-08-20 23:22:16 UTC

Contents

calculateMemoryDemandGPU	3
Cosine_Distance	4
distance-conventions	5
DistanceDistributionAnalysis	6
DistanceDistributions	10
DistanceMatrix	13
DistanceOneToAll	20
DistanceProperties	21
DTW_Distance	23
EndresSchindelin_Distance	24
endresSchindelin_Distribution	25
endresSchindelin_TwoSamplePDF	26
EuclideanGPU_Distance	28
EuclideanMulticore_Distance	30
fast_dip_stat	30
fast_dip_test	32
fast_dip_test_matrix	34
Fractional_Distance	36
Gini_Distance	37
Hearingloss_N109	38
Hellinger_Distance	38
Hepta	39
IntraClusterBoundaryCounts	40
Jaccard_Distance	42
KNearestNeighborsFromDistance	43
Kullback_Leibler_div	44
Mahalanobis_Distance	45
Minkowski_Distance	47
MSMD_Distance	49
NearestNeighborIndex	50
SharedNeighbor_Distance	51
Tfidf_Distance	53
ToroidalEuclideanDistanceOneToAll	55

calculateMemoryDemandGPU 3

ToroidalEuclidean_Distance	56
TransformSimilarity2MetricDistance	57
TWED_Distance	57
VariablePrecision	59
Wasserstein_Distance	60

Index 62

`calculateMemoryDemandGPU`
Plan Device-Memory Batches for Weighted Euclidean Distances

Description

Selects one of three OpenCL execution layouts and calculates the corresponding batch sizes from the number of rows, number of columns, and a device-memory budget.

Usage

```
calculateMemoryDemandGPU(n, d, mem = 2)
```

Arguments

<code>n</code>	One positive integer: the number of observations (rows).
<code>d</code>	One positive integer: the number of features (columns).
<code>mem</code>	One finite positive number giving the memory budget in GiB.

Details

The estimate conservatively uses eight bytes per input, weight, and output value.

Version 0 The complete input and output matrix fit in the budget.

Version 1 The complete input fits, while output columns are processed in batches.

Version 2 Two input blocks and their output block are processed at a time.

The calculation is a planning estimate. Device-specific limits, driver memory, and other allocations can still prevent an OpenCL allocation. With `backend = "auto"`, `EuclideanGPU_Distance()` catches such failures and uses its CPU backend.

Value

A list with components:

<code>maxBatchSize</code>	Maximum planned rows or output columns per batch.
<code>minNrBatches</code>	Number of planned batches.
<code>batchSizes</code>	Integer vector containing each batch size.
<code>Version</code>	Integer execution-layout identifier: 0, 1, or 2.

Author(s)

Luca Brinkmann, with portability revisions by Michael Thrun

See Also

[EuclideanGPU_Distance](#)

Examples

```
info = calculateMemoryDemandGPU(70000, 784, 6)
```

Cosine_Distance

Cosine Distance Between Rows

Description

Computes $1 - s_{ij}$, where s_{ij} is the cosine similarity between rows i and j of a numerical matrix.

Usage

```
Cosine_Distance(Data)
```

Arguments

Data [1:n,1:d] numerical matrix with n cases, d variables

Details

The implementation uses a normalized cross-product and returns a full matrix. Two zero rows are treated as identical and have distance zero. A zero row and a non-zero row have similarity zero and distance one. Similarities are clamped to the interval from -1 to 1 to remove floating-point overshoot.

Value

[1:n,1:n] numerical matrix of distances.

Note

The cosine distance is calculated from the cosine similarity as $d(i, j) = 1 - s(i, j)$, where s is the cosine similarity and d is the cosine distance.

Author(s)

Michael Thrun

Examples

```
X = rbind(c(1, 0), c(0, 1), c(0, 0), c(0, 0))
D = Cosine_Distance(X)
stopifnot(D[3, 4] == 0, all(is.finite(D)))
```

distance-conventions *Distance, Metric, and Dissimilarity Conventions in BIDs*

Description

Clarifies how the package uses the terms distance, metric, pseudometric, and dissimilarity.

Details

A mathematical metric d satisfies non-negativity, symmetry, identity of indiscernibles, and the triangle inequality. A pseudometric may assign zero to distinct objects. A general dissimilarity need not satisfy symmetry or the triangle inequality.

BIDistances uses *distance* as a broad software term because many R interfaces and publications follow this convention. The name of a function therefore does not by itself establish that all metric axioms hold for every admissible input and parameter value. Examples include:

- Euclidean and non-negative weighted Euclidean distances are metrics after zero-weighted coordinates are removed, provided distinct remaining feature vectors cannot collapse to the same representation.
- Weighted Minkowski distances are metrics for $p \geq 1$ when all coordinate weights are positive. Zero coordinate weights can collapse distinct vectors, yielding a pseudometric. In `Minkowski_Distance()`, $p = \infty$ is available with unit weights. Fractional choices $0 < p < 1$ generally violate the triangle inequality and are dissimilarities rather than metrics.
- The square-root Endres–Schindelin construction is a metric on probability distributions. A finite-grid density estimate is a numerical approximation to that metric.
- A distance formed by the absolute difference of scalar tf–idf weights can be a pseudometric on genes: different genes may receive the same scalar weight and consequently have zero distance.
- Correlation-, cosine-, and application-specific transformations require their own domain assumptions and should not be assumed to satisfy all metric axioms solely from their names.

Users should match downstream algorithms to the mathematical properties they require. In particular, algorithms or validation indices that rely on the triangle inequality should not be applied automatically to a general dissimilarity.

References

- Endres, D. M. and Schindelin, J. E. (2003). A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7), 1858–1860. doi:10.1109/TIT.2003.813506.
- Stier, Q. and Thrun, M. C. (2023). Deriving homogeneous subsets from gene sets by exploiting the Gene Ontology. *Informatica*, 34(2), 357–386. doi:10.15388/23INFOR517.

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space, Database Theory, ICDT 2001, pp. 420-434, DOI: 10.1007/3-540-44503-X_27.

See Also

[DistanceMatrix](#), [Fractional_Distance](#), [endresSchindelin_Distribution](#), [Tfidf_Distance](#), [EuclideanGPU_Distance](#), [Minkowski_Distance](#)

DistanceDistributionAnalysis

Analyze and Select Distance Distributions

Description

Analyzes already calculated pairwise distance distributions and selects the distance whose one-dimensional distribution shows the strongest suitable bimodality.

No distances are calculated from raw observations by this function.

Usage

```
DistanceDistributionAnalysis(
  Distances,
  PlotIt = FALSE,
  PlotSampleSize = 5e3
)
```

Arguments

Distances	Distance candidates in one of the following forms: • a <code>stats::dist</code> object; • a named list of square numeric distance matrices and/or <code>stats::dist</code> objects; • a numeric matrix or data frame in which every column is one distance candidate and every row represents the same upper-triangle observation pair across all columns. For matrix input, the number of rows must equal $n(n - 1)/2$ for an integer number of observations n.
PlotIt	Logical scalar. If TRUE, the ordered distance-distribution plot is generated, printed, and returned. Default is FALSE.
PlotSampleSize	Positive finite numeric scalar giving the number of pairwise values used by <code>DataVisualizations::MDplot()</code>. It is converted to an integer by applying <code>floor()</code>. Default is 5e3.

Details

Selection criteria

The function applies the established distance-distribution criteria used by `DistanceDistributions()`.

A candidate must satisfy all of the following conditions:

- more than 100 unique distance values;
- kurtosis below 10;
- Hartigan dip-test p-value below 0.05;
- bimodality amplitude of at least 0.3.

Among all candidates satisfying these conditions, the candidate with the largest bimodality amplitude is selected.

Kurtosis is calculated with `e1071::kurtosis()`. For a distance vector with at most 72000 values, modality is tested with the established `dipTest::dip.test()` implementation. For strictly more than 72000 values, all candidates that passed the unique-value and kurtosis criteria are sent together to `fast_dip_test_matrix()` with `p.value.method = "asymptotic"`. The full distance vectors are used; no downsampling is performed.

The large-sample batch copies and sorts one selected distance column at a time inside C++ and reuses its linear workspace for subsequent columns. This avoids one native-call transition and one separately allocated work array per large distance candidate. Its p-value calibration uses the scaled $n = 72000$ critical-value row as an approximation to the limiting distribution of $\sqrt{n}D_n$.

Both dip-test paths store the observed Hartigan dip statistic in `DipStatistic`. The bimodality amplitude is calculated independently with `DataVisualizations::BimodalityAmplitude()`.

Errors during kurtosis, dip-test, or bimodality calculations are converted to conservative values that cause the affected distance to fail the corresponding criterion.

Status values

Each candidate receives one of the following status strings:

- "selected": candidate selected through the regular criteria;
- "selected_by_bimodality_amplitude": selected through the documented fallback when every analyzable dip test is non-significant;
- "candidate": passes all regular criteria but ranks below the selected candidate or candidates;
- "bimodality_below_threshold": bimodality amplitude below 0.3;
- "dip_not_significant": dip-test p-value at least 0.05;
- "bimodality_calculation_failed": bimodality amplitude could not be calculated as one finite value;
- "excessive_kurtosis": kurtosis at least 10;
- "too_few_unique_values": at most 100 unique values.

The returned statistics and distance columns are ordered by status, then by decreasing finite bimodality amplitude, increasing dip-test p-value, increasing kurtosis, decreasing number of unique values, and distance name.

List input

A single `dist` object is treated as a one-element distance list.

For list input, every candidate is converted to a matrix and must be finite, numeric, square, at least 2×2 , symmetric, non-negative within floating-point tolerance, and have a zero diagonal within tolerance.

All matrices must have identical dimensions. If observation labels are present, row and column labels within each matrix must be identical, all candidates must either have labels or not have labels, and all labeled matrices must use the same observation order.

Candidate names are made unique. Missing names are replaced by "Distance_<index>".

Upper-triangle entries are extracted in a common order and used for the distribution analysis.

Matrix input

For numeric matrix or data-frame input, every column represents a distance candidate and every row must correspond to the same unordered observation pair in every column.

The number of rows must form a complete upper triangle, that is, $m = n(n - 1)/2$ for an integer n . Because the original observation ordering cannot be reconstructed from this representation, the function can return only the selected name and ordered vectors, not a selected square matrix.

Non-finite or materially negative values are rejected. Tiny negative values within floating-point tolerance are set to zero.

Plotting

When `PlotIt = TRUE`, `DataVisualizations::MDplot()` is applied to the ordered distance vectors with column-wise ordering and percentalized scaling. The plot title indicates whether a suitable distance was found.

Package **ggplot2** is required only when plotting is requested.

Value

A list with components:

<code>DistanceMatrix</code>	Selected square numeric distance matrix when <code>Distances</code> was supplied as a list of complete distance matrices or <code>dist</code> objects. It is <code>NULL</code> for matrix input, when no suitable distance is found, or when no full matrix representation is available.
<code>DistanceChoice</code>	Character scalar or vector naming the selected distance or tied distances, or "Not Found" when no candidate satisfies the applicable selection criteria.
<code>OrderedDistances</code>	Numeric matrix with one row per unordered observation pair and one column per candidate distance. Columns are ordered according to candidate status and, within status, decreasing bimodality amplitude.
<code>ggobject</code>	Plot object created by <code>DataVisualizations::MDplot()</code> when <code>PlotIt = TRUE</code> ; otherwise <code>NULL</code> . Plot-generation errors are converted to warnings and result in <code>NULL</code> .
<code>SelectionStatistics</code>	Data frame with one row per candidate and columns <code>Distance</code> , <code>UniqueValues</code> , <code>Kurtosis</code> , <code>DipStatistic</code> , <code>DipPValue</code> , <code>BimodalityAmplitude</code> , <code>Selected</code> , and <code>Status</code> . <code>DipStatistic</code> and <code>DipPValue</code> are <code>NA</code> when the candidate does not reach the dip-test stage.

Note

The packages **DataVisualizations**, **dipTest**, and **e1071** are required for every analysis. The large-sample dip implementation is compiled into **BDistances**; no separate **fastdip** installation is required. The package **ggplot2** is additionally required when `PlotIt = TRUE`.

For matrix input, vectors should be produced in a consistent upper-triangle order. In particular, vectors returned by `DistanceMatrix(..., outputisvector = TRUE)` follow the intended format.

Author(s)

Michael Thrun

References

Thrun, M. C. (2021). The Exploitation of Distance Distributions for Clustering. *International Journal of Computational Intelligence and Applications*, **20**(3), 2150016. doi: 10.1142/S1469026821500164.

See Also

[DistanceDistributions](#), [fast_dip_test_matrix](#), [fast_dip_test](#)

Examples

```
## Not run:
set.seed(1)
X <- matrix(rnorm(80), ncol = 4)

D1 <- as.matrix(stats::dist(X))
D2 <- as.matrix(stats::dist(X, method = "manhattan"))

result <- DistanceDistributionAnalysis(
  Distances = list(
    Euclidean = D1,
    Manhattan = D2
  )
)

result$DistanceChoice
result$SelectionStatistics

# Matrix input: columns must describe the same upper-triangle pairs
upper <- upper.tri(D1, diag = FALSE)
DistanceFeatures <- cbind(
  Euclidean = D1[upper],
  Manhattan = D2[upper]
)

result2 <- DistanceDistributionAnalysis(DistanceFeatures)

## End(Not run)
```

Description

Calculates multiple pairwise distance measures between the rows of a numeric data matrix, analyzes their one-dimensional distance distributions with `DistanceDistributionAnalysis()`, and returns the distance whose distribution shows the strongest suitable bimodality.

The function delegates distance calculation to `DistanceMatrix()` and separates candidate generation from distribution-based distance selection.

Usage

```
DistanceDistributions(
  Data,
  DistanceMethods = c(
    "bhjattacharyya", "bray", "canberra", "chord", "divergence",
    "euclidean", "minkowski", "geodesic", "hellinger", "kullback",
    "manhattan", "maximum", "soergel", "wave", "whittaker",
    "Cosine_Distance", "pearsond"
  ),
  P = c(0.1, 0.5, 4, 10, 100),
  PlotIt = FALSE,
  PlotSampleSize = 5e3
)
```

Arguments

Data	Numeric matrix with n observations in rows and d variables in columns. At least two rows and one column are required. All values must be finite.
DistanceMethods	Non-empty character vector specifying distance methods to evaluate. Ordinary entries are passed directly to <code>DistanceMatrix()</code>. Methods such as "Cosine_Distance", "pearsond", "mahalanobis", and "podani" can therefore be requested directly when supported by <code>DistanceMatrix()</code>. The entries "minkowski" and "Minkowski" are intentionally case-sensitive and select different computational routes; see Details.
P	Numeric vector of positive Minkowski exponents. Every value is evaluated for each requested Minkowski route. Finite positive values and Inf are allowed. For the GPU-aware "Minkowski" or "Minkowski_Distance" route, all values must be at least 1 or Inf. Default is <code>c(0.1, 0.5, 4, 10, 100)</code>.
PlotIt	Logical scalar. If TRUE, the distance-distribution plot generated by <code>DistanceDistributionAnalysis()</code> is printed. Default is FALSE.
PlotSampleSize	Positive finite numeric scalar giving the number of pairwise distance values used by <code>DataVisualizations::MDplot()</code> when plotting. Default is 5e3.

Details

Candidate calculation

`DistanceDistributions()` compares distances between rows of `Data`. Every candidate is first calculated as a vector containing all $n(n-1)/2$ upper-triangle pairwise distances. A candidate is omitted with a warning if calculation fails, does not contain all row pairs, contains non-finite values, or contains materially negative distances.

Very small negative values within a floating-point tolerance are set to zero.

If none of the requested candidates can be calculated successfully, the function stops.

Minkowski routes

Two intentionally different Minkowski routes are supported:

- "minkowski" calls `DistanceMatrix()` with `method = "minkowski"` and `GPU = FALSE`. This corresponds to the `parallelDist`-based route. Positive exponents below 1 are permitted by this wrapper. When $P = \text{Inf}$, this route uses `method = "maximum"`.
- "Minkowski" calls `DistanceMatrix()` with `method = "Minkowski_Distance"` and `GPU = TRUE`. The explicit spelling "Minkowski_Distance" is equivalent. This route requires $P \geq 1$ or Inf .

Candidate names encode the exponent as "minkowski_p_<p>" or "Minkowski_p_<p>".

The exponent must be supplied through `P`. Method strings that already encode "_p_" are rejected to prevent two competing sources for the Minkowski exponent.

Case variants that could ambiguously refer to the two Minkowski routes are also rejected. Use exactly "minkowski" for the `parallelDist` route or exactly "Minkowski" for the GPU-aware route.

Distribution analysis

All successfully calculated upper-triangle vectors are combined column-wise and passed to `DistanceDistributionAnalysis`. That function evaluates the shape of each distance distribution and chooses the strongest suitable bimodal candidate. It retains `dip.test::dip.test()` for vectors with at most 72000 pairwise values and automatically uses the integrated batch `fast_dip_test_matrix()` implementation above that size. Both paths return the observed dip statistic in `SelectionStatistics$DipStatistic`.

If a distance is selected, its complete square matrix is calculated once more using exactly the same method, GPU flag, and Minkowski exponent specification. The reconstructed result must be a finite numeric $n \times n$ matrix; otherwise a warning is issued and `DistanceMatrix` is returned as `NULL`.

The maximum distance corresponds to the Chebyshev or L_∞ norm, while Manhattan distance corresponds to the L_1 norm.

Value

A list returned by `DistanceDistributionAnalysis()` with components:

- | | |
|-----------------------------|--|
| <code>DistanceMatrix</code> | Numeric $n \times n$ matrix containing the selected distance measure, reconstructed with the same <code>DistanceMatrix()</code> specification used for candidate generation. It is <code>NULL</code> if no suitable distance is selected or if reconstruction fails. |
| <code>DistanceChoice</code> | Character scalar naming the selected candidate, or "Not Found" when no candidate satisfies the selection criteria. |

OrderedDistances

Numeric matrix with one row per unordered observation pair and one column per successfully calculated distance candidate. Columns are ordered from more suitable to less suitable according to `DistanceDistributionAnalysis()`.

ggobject

Plot object returned by `DataVisualizations::MDplot()` when `PlotIt = TRUE`; otherwise NULL.

SelectionStatistics

Data frame returned by `DistanceDistributionAnalysis()` containing the number of unique values, kurtosis, Hartigan dip statistic, dip-test p-value, bimodality amplitude, selection flag, and status for every successfully calculated candidate.

Note

A method that returns distances between columns rather than rows cannot be combined with this analysis because all candidate vectors must describe the same unordered row pairs in the same order.

Author(s)

Michael Thrun

References

Thrun, M. C. (2021). The Exploitation of Distance Distributions for Clustering. *International Journal of Computational Intelligence and Applications*, **20**(3), 2150016. doi: 10.1142/S1469026821500164.

See Also

[DistanceDistributionAnalysis](#)

Examples

```
## Not run:
set.seed(1)
Data <- rbind(
  matrix(rnorm(100, 0, 1), ncol = 2),
  matrix(rnorm(100, 5, 1), ncol = 2)
)

result <- DistanceDistributions(
  Data = Data,
  DistanceMethods = c("euclidean", "manhattan", "minkowski"),
  P = c(1, 2, 4),
  PlotIt = FALSE
)

result$DistanceChoice
result$SelectionStatistics

# Request the GPU-aware Minkowski_Distance route:
result2 <- DistanceDistributions(
  Data = Data,
```

```

    DistanceMethods = c("euclidean", "Minkowski"),
    P = c(1, 2, Inf)
  )

  ## End(Not run)

```

DistanceMatrix

Pairwise Distance, Dissimilarity, or Similarity Matrix

Description

Computes pairwise values for a data matrix. Most methods compare cases in the rows of X . The `BIDistances` routes `"DTW_Distance"` and `"MSMD_Distance"` compare time series in the columns, and `"EndresSchindelin_Distance"` compares variables in the columns. **parallelDist** is tried first for ordinary numerical methods, and **philentropy** is used only when **parallelDist** reports that the method name is invalid. Mixed-data calculations use an explicit **manydist** backend.

Usage

```

DistanceMatrix(
  X,
  method = "euclidean",
  dim = 2,
  outputisvector = FALSE,
  GPU = FALSE,
  ...
)

```

Arguments

X [1:n,1:d] numerical matrix with n cases, d variables. A data frame is also accepted for `method = "manydist"` so that mixed numerical and factor variable types are preserved. Missing values are not allowed. For `"DTW_Distance"` and `"MSMD_Distance"`, every column is one time series and the row order is the temporal order.

method One non-empty character string. Matching is case-insensitive.

The canonical `BIDistances` method names are `"Gini_Distance"`, `"EndresSchindelin_Distance"`, `"Fractional_Distance"`, `"Minkowski_Distance"`, `"MSMD_Distance"`, `"SharedNeighbor_Distance"`, `"DTW_Distance"`, `"Tfidf_Distance"`, `"Wasserstein_Distance"`, `"Jaccard_Distance"`, `"Cosine_Distance"`, and `"Mahalanobis_Distance"`. These canonical names are matched case-insensitively. `"Tfidf_dist"` remains an alias of `"Tfidf_Distance"`.

The correlation routes are `"pearsond"`, `"spearmand"`, `"kendalld"`, `"pearsonm"`, `"spearmanm"`, and `"kendallm"`.

`"TWED_Distance"` is not dispatched because it requires explicit `Time1` and `Time2` vectors. Call `TWED_Distance()` directly.

Method labels of the form `"minkowski_p_<p>"`, such as `"minkowski_p_0.5"` and `"minkowski_p_4"`, are aliases for `method = "minkowski"` with the encoded

positive exponent p . These are the labels returned by `DistanceDistributions()`. A separate p argument must not be supplied with an encoded alias.

The wrapper-specific route "sqeuclidean" squares a **parallelDist** Euclidean matrix. On the CPU, this route accepts applicable **parallelDist** arguments such as threads.

Every other ordinary method name is first passed to `parallelDist::parDist()`. Only its "Invalid distance method" error triggers a lookup in `philentropy::getDistMethods()`. Consequently, a name implemented by both packages always selects the **parallelDist** definition.

Use "manydist" or "mdist" for mixed-data calculations and select the many-dist specification through ..., for example `preset = "gower"` or `preset = "custom"`, `method_cat = "matching"`.

dim	One finite positive number. It is used as the default Minkowski exponent p , the Wasserstein order, or the fractional-distance exponent. A Minkowski, Wasserstein, or fractional p supplied through ... overrides dim.
outputisvector	Logical scalar. If false, return the full square matrix. If true, return <code>D[upper.tri(D)]</code> , preserving the historical <code>BIDistances</code> ordering. If the result is asymmetric, only its upper triangle is returned and a warning is issued.
GPU	Logical scalar. If true, use <code>EuclideanGPU_Distance()</code> for "euclidean" and "sqeuclidean". For "Minkowski_Distance", TRUE selects backend = "auto" and FALSE selects backend = "parallelDist" when backend is not supplied through An explicitly supplied Minkowski backend takes precedence. Other methods use the CPU.
...	Uniquely named arguments forwarded to the backend selected by method. Arguments are validated for every <code>BIDistances</code> -specific route. Supply only the arguments described in the backend-specific sections below. The Euclidean GPU backend accepts only <code>Weights</code> , <code>Mem</code> , <code>ctx</code> , <code>backend</code> , and <code>threads</code> . The canonical Minkowski route accepts p , <code>Weights</code> , <code>backend</code> , <code>Mem</code> , <code>ctx</code> , and <code>threads</code> .

Details

The aliases "cityblock" and "manhatten" map to "manhattan"; "chebychev" and "chebyshev" map to "maximum"; "braycur" and "braycurtis" map to "bray"; and "squared_euclidean" maps to "sqeuclidean". `BIDistances`-specific implementations must be requested with their canonical "_Distance" names, except for the retained "Tfidf_dist" alias.

The ordinary method "minkowski" remains a direct **parallelDist** route. The canonical "Minkowski_Distance" route adds coordinate weights and explicit "parallelDist", "multicore", "opencl", and "auto" backends.

Plain overlapping names such as "cosine", "hellinger", "mahalanobis", and "tanimoto" use **parallelDist**. "Cosine_Distance" and "Mahalanobis_Distance" select the implementations provided by `BIDistances`. Plain "gower" uses the numerical **philentropy** definition; mixed-data Gower is requested with `method = "manydist"`, `preset = "gower"`.

The **philentropy** registry includes both distances and similarities. Values are returned unchanged. Some methods are asymmetric, and many are intended for non-negative probability or count vectors.

Value

[1:n,1:n] numerical matrix of distances. When `outputisvector = TRUE`, the upper triangle is returned as a numerical vector. No `dist` object is returned. The time-series routes and `EndresSchindelin_Distance` compare columns; the other routes compare cases in rows.

Rules for additional arguments

Every argument in ... must have a unique, non-empty name. `x`, `X`, `Data`, and `method` are controlled by `DistanceMatrix()` and are rejected when supplied through ...

Backend selection happens before the additional arguments are interpreted. In particular, a method name implemented by both **parallelDist** and **philentropy** selects **parallelDist**. Supplying a **philentropy**-only argument with such a name does not force the **philentropy** backend; it normally produces a **parallelDist** argument error. Use `threads` for **parallelDist** and `num.threads` for **philentropy**; the two names are not interchangeable.

Additional arguments for the GPU backend

When `GPU = TRUE` and the method is "euclidean" or "sqeuclidean", the following names may be supplied through ...: `Weights`, `Mem`, `ctx`, `backend`, and `threads`. They have the meanings documented in [EuclideanGPU_Distance](#). All other names are rejected before dispatch. The square-Euclidean route squares the matrix returned by that function.

For `method = "Minkowski_Distance"`, GPU selection is controlled by the canonical route described below. Its OpenCL `p = 2` case delegates to the existing Euclidean GPU implementation; other exponents use separate Minkowski kernels.

BIDistances-specific methods

The following canonical method names dispatch directly to functions in this package. Legacy `BIDistances` function names are rejected, except that "Tfidf_dist" remains an alias of "Tfidf_Distance".

"Gini_Distance" Calls `Gini_Distance(Data)`. No additional argument is accepted.

"Cosine_Distance" Calls `Cosine_Distance(Data)`. No additional argument is accepted.

"Mahalanobis_Distance" Calls `Mahalanobis_Distance(X, cov, inverted)`. The positive-definite `cov` argument is required; `inverted` is optional. This canonical route returns squared generalized Mahalanobis distances.

"EndresSchindelin_Distance" Calls `EndresSchindelin_Distance(Data, ncores)`. The columns are compared as samples of variables. The optional argument is `ncores`.

"Fractional_Distance" Calls `Fractional_Distance(Data, p)`. Supply `p` through ...; when it is omitted, `dim` is used.

"Minkowski_Distance" Calls `Minkowski_Distance()` and compares rows. Supported arguments are `p`, `Weights`, `backend`, `Mem`, `ctx`, and `threads`. When `p` is omitted, `dim` is used. When `backend` is omitted, `GPU = TRUE` selects "auto" and `GPU = FALSE` selects "parallelDist". The route accepts only $p \geq 1$ or `Inf`; use `Fractional_Distance()` for $0 < p < 1$.

"MSMD_Distance" Each column is one univariate time series and the row order is the temporal order. No time variable should be given. The required argument is `ParameterC`.

"SharedNeighbor_Distance" Calls `SharedNeighbor_Distance()`. Supported arguments are `k`, `NThreads`, `ComputationInR`, and `verbose`.

"DTW_Distance" Each column is one time series and the row order is the temporal order. No time variable should be given. DistanceFunction and further arguments of that function are accepted. Because method names the wrapper route, use DistanceFunctionMethod to pass a method name to DistanceFunction.

"Tfidf_Distance" Calls Tfidf_Distance() and returns its Distance component. The optional argument is tf_fun.

"Wasserstein_Distance" Calls Wasserstein_Distance(). Supported arguments are p and InverseWeighting; when p is omitted, dim is used.

"Jaccard_Distance" Calls Jaccard_Distance(Data). No additional argument is accepted.

TWED_Distance() requires explicit time vectors and is intentionally not included. Calling DistanceMatrix(..., method = "TWED_Distance") gives an explanatory error.

Additional arguments for parallelDist

The following arguments apply when the selected method is implemented by parallelDist::parDist():

threads Number of CPU threads. With NULL, **parallelDist** uses the maximum number available on the system.

p Positive Minkowski exponent, used only with method = "minkowski". When omitted, DistanceMatrix() supplies p = dim.

cov Covariance matrix used only with method = "mahalanobis". When omitted, **parallelDist** estimates the covariance matrix from X.

inverted Logical value used only with Mahalanobis distance. If true, cov is interpreted as an inverse covariance matrix.

window.size Optional integer width of the Sakoe–Chiba window used only with method = "dtw".

norm.method Optional DTW normalization. Supported values are "path.length", "n", and "n+m".

step.pattern DTW step pattern. The backend default is "symmetric1". See [parDist](#) for the complete set of supported patterns.

func An external pointer to a compiled C++ distance function, used only with method = "custom". See [parDist](#) for the required function signature and a compilation example.

diag, upper Printing flags attached to the temporary dist object. Because DistanceMatrix() immediately converts that object to a full matrix, these arguments do not change the returned value and should normally be omitted.

The CPU routes method = "sqeuclidean" and method = "fagerdissimilarity" also use this backend. Applicable arguments such as threads are therefore forwarded to the underlying Euclidean or Fager calculation.

All other method-specific semantics, including data-domain restrictions, are those of parallelDist::parDist().

Additional arguments for philentropy

These arguments apply only when **parallelDist** rejects the method name and the name is present in philentropy::getDistMethods(). For methods that compare probability distributions, rows of X should be non-negative probability vectors, or non-negative count vectors together with est.prob = "empirical".

`test.na` Logical value controlling a second missing-value check inside **philentropy**. If omitted, `DistanceMatrix()` sets it to `FALSE`, because `X` has already been checked.

`unit` Logarithm used by log-dependent methods. Valid values are `"log"` for the natural logarithm, `"log2"`, and `"log10"`. The backend default is `"log"`.

`epsilon` Small positive replacement used by applicable methods in otherwise undefined zero-denominator cases. The backend default is `1e-5`. Results from zero-heavy distributions can be sensitive to this value, so it should be chosen for the scale and sparsity of the data.

`est.prob` Probability estimation from row-wise counts. The default `NULL` uses rows as supplied. `"empirical"` divides each row by its row sum before the distance is calculated.

`mute.message` Logical value that suppresses messages from `philentropy::distance()`. If omitted, `DistanceMatrix()` sets it to `TRUE`.

`num.threads` Number of worker threads used by **philentropy**. With `NULL`, the backend uses `RCPP_PARALLEL_NUM_THREADS`, or two threads when that environment variable is unset.

`use.row.names` Controlled by the wrapper and always set to `FALSE`. The row names of `X`, when present, are restored on the final matrix.

`as.dist.obj` Controlled by the wrapper and always set to `FALSE`, because `DistanceMatrix()` requires a full matrix from the backend.

`diag, upper` These only control printing when `as.dist.obj = TRUE`. They therefore do not affect the matrix returned by `DistanceMatrix()` and should normally be omitted.

The `p` argument of `philentropy::distance()` is not normally reachable through this wrapper: `"minkowski"` is implemented by **parallelDist** and is therefore dispatched there first.

Additional arguments for **manydist**

These arguments apply only with `method = "manydist"` and are forwarded to `manydist::mdist()`. Character and logical predictor columns are converted to factors before dispatch.

`preset` Predefined distance specification. The default is `"custom"`. Current choices include `"gower"`, `"unbiased_dependent"`, `"u_dep"`, `"u_indep"`, `"u_mix"`, `"hl"`, `"gudmm"`, `"dkss"`, `"mod_gower"`, and `"euclidean"`. With a non-custom preset, the preset controls `method_cat`, `method_num`, `commensurable`, and `interaction`; values supplied for those arguments are ignored by **manydist**.

`response` Optional response column for response-aware categorical dissimilarities. In this wrapper it must be supplied as a character column name, for example `response = "class"`, because the contents of `...` are evaluated before `manydist::mdist()` is called. **manydist** removes the response column from the predictor set.

`method_cat` Categorical-variable dissimilarity used with `preset = "custom"`. Common choices are `"matching"` and `"tvd"`. Depending on the selected specification, this can be one method for all categorical variables or a method vector aligned with the categorical columns. Use `manydist::all_dist_method_specs()` to inspect the methods in the installed package.

`method_num` Numerical preprocessing used with `preset = "custom"`. Choices are `"none"` for no preprocessing, `"std"` for standard-deviation scaling, `"range"` for range scaling, `"robust"` for interquartile-range scaling, and `"pc_scores"` for principal-component score scaling. The default is `"std"`.

- commensurable** Logical value. When true and supported by the selected specification, variable-wise dissimilarities are scaled so that their average contributions to the overall dissimilarity are equal to one. The default is TRUE for the custom specification.
- ncomp** Positive integer or NULL; number of principal components retained when `method_num = "pc_scores"`. With NULL, all available components are used unless `threshold` is supplied and supported.
- threshold** Numeric value between zero and one, or NULL; optional cumulative-variance threshold used with `method_num = "pc_scores"`.
- interaction** Logical value. If true, add an interaction-aware continuous–categorical component based on local predictive separability. The default is FALSE.
- prop_nn** Proportion between zero and one giving the nearest-neighbour fraction used only when `interaction = TRUE`. The default is 0.1.
- score** Interaction score used only when `interaction = TRUE`. Available values include "ba" for balanced accuracy and "logloss". The default is "ba".
- decision** Decision rule used when `interaction = TRUE` and `score = "ba"`. The default is "prior_corrected"; consult [mdist](#) for supported rules.
- gower_average** Used only with `preset = "gower"`. The default TRUE averages over active variables and matches the usual Gower scale. FALSE returns the sum of the variable-wise Gower contributions.
- new_data** Not supported by `DistanceMatrix()`. Although `manydist::mdist()` can calculate rectangular new-data-to-training dissimilarities, this wrapper always returns a square within-X matrix.

The installed **manydist** documentation is authoritative for preset and method registries, which may grow in later releases.

Similarity orientation

BIDistances uses the word *similarity* for a distance-like pairwise matrix whose small values indicate dissimilar objects and whose large values indicate similar objects. Thus a similarity has the opposite ordering from a distance, but it is stored in the same full square matrix form. A finite symmetric similarity already scaled to $[0, 1]$ with diagonal one can be transformed explicitly with `TransformSimilarity2MetricDistance()`; other similarity scales must first be converted to that domain.

Author(s)

Michael Thrun

References

- Sneath, P. H. A. (1957) Some thoughts on bacterial classification. *Journal of General Microbiology* 17, pages 184-200.
- Leydesdorff, L. (2005) Similarity Measures, Author Cocitation Analysis, and Information Theory. In: *JASIST* 56(7), pp.769-772.
- Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

- Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979) Multivariate Analysis. Academic Press.
- Borg, I. and Groenen, P. (1997) Modern Multidimensional Scaling. Theory and Applications. Springer.
- Mahalanobis, P. C. (1936) On the generalized distance in statistics. Proceedings of The National Institute of Sciences of India, 12:49-55.
- van de Velden, M., Iodice D'Enza, A., Markos, A., & Cavicchia, C. (2026). Unbiased Mixed-Variable Distance. Journal of Computational and Graphical Statistics, 1–12. DOI, 10.1080/10618600.2026.2680181

See Also

[parDist](#), [getDistMethods](#), [distance](#), [mdist](#), [Gini_Distance](#), [EndresSchindelin_Distance](#), [Fractional_Distance](#), [Minkowski_Distance](#), [MSMD_Distance](#), [SharedNeighbor_Distance](#), [DTW_Distance](#), [Tfidf_Distance](#), [Wasserstein_Distance](#), and [Jaccard_Distance](#).

Examples

```
X = rbind(a = c(1, 2, 3), b = c(2, 2, 4), c = c(4, 1, 2))
DistanceMatrix(X, method = "euclidean")
DistanceMatrix(X, method = "minkowski", p = 3)

# Method-specific parallelDist arguments
DistanceMatrix(X, method = "mahalanobis", cov = diag(ncol(X)))
DistanceMatrix(X, method = "dtw", window.size = 1,
               norm.method = "path.length")

# BIDistances-specific routes
DistanceMatrix(X, method = "Fractional_Distance", p = 0.5)
DistanceMatrix(
  X,
  method = "Minkowski_Distance",
  p = 3,
  Weights = c(1, 2, 0.5),
  backend = "multicore",
  threads = 2
)
DistanceMatrix(X, method = "SharedNeighbor_Distance", k = 1,
               NThreads = 1, ComputationInR = TRUE)

TimeSeries = cbind(Series1 = 1:5, Series2 = c(1, 2, 3, 4, 7))
DistanceMatrix(TimeSeries, method = "MSMD_Distance", ParameterC = 1)

Gene2G0 = rbind(
  Gene1 = c(1, 0, 2),
  Gene2 = c(0, 1, 1),
  Gene3 = c(1, 1, 0)
)
DistanceMatrix(Gene2G0, method = "Tfidf_Distance")

if (requireNamespace("philentropy", quietly = TRUE)) {
  counts = rbind(a = c(1, 4, 5), b = c(2, 3, 5), c = c(6, 2, 2))
  DistanceMatrix(
```

```

        counts,
        method = "jensen-shannon",
        est.prob = "empirical",
        unit = "log2",
        epsilon = 1e-8
    )
}

if (requireNamespace("manydist", quietly = TRUE)) {
    mixed = data.frame(
        value = c(1, 3, 2),
        group = factor(c("a", "b", "a"))
    )
    DistanceMatrix(
        mixed,
        method = "manydist",
        preset = "gower",
        gower_average = TRUE
    )
    DistanceMatrix(
        mixed,
        method = "manydist",
        preset = "custom",
        method_cat = "matching",
        method_num = "robust",
        commensurable = TRUE
    )

    # In this wrapper, response must be a character column name:
    # DistanceMatrix(mixed_with_class, method = "manydist",
    #                 preset = "custom", method_cat = "tvd",
    #                 response = "class")
}

```

DistanceOneToAll

Distances from One Point to All Rows of a Matrix

Description

Calculates the distance from a numerical vector to every row of a numerical matrix and returns the indices of the nearest rows.

Usage

```

DistanceOneToAll(
  X,
  Data,
  SelectFeatures,
  method = "euclidean",
  p = 2,

```

```
knn = 1,
GPU = FALSE
)
```

Arguments

X	A numerical vector with one value per column of Data.
Data	[1:n,1:d] numerical matrix with n cases, d variables
SelectFeatures	Optional logical vector of length ncol(Data). Numeric vectors containing only zero and one are also accepted. At least one column must be selected.
method	Distance method forwarded to <code>parallelDist::parDist()</code> . The default is "euclidean".
p	One finite positive number used for method = "minkowski".
knn	One positive integer. At most nrow(Data) nearest-neighbour indices are returned.
GPU	Logical scalar. When true and method = "euclidean", use <code>EuclideanGPU_Distance()</code> ; its automatic CPU fallback remains active. For other methods a warning is issued and the CPU is used.

Value

A list with:

distToAll	Numeric vector containing one distance for every row of Data, in row order.
KNN	Integer vector containing the row indices of the nearest points.

Author(s)

Michael Thrun

Examples

```
data(Hepta)
result = DistanceOneToAll(Hepta$Data[1, ], Hepta$Data)
stopifnot(length(result$distToAll) == nrow(Hepta$Data))
```

DistanceProperties	<i>Mathematical Property Registry for BIDistances Methods</i>
--------------------	---

Description

Returns a machine-readable summary of the mathematical properties of native BIDistances distance, dissimilarity, and divergence constructions.

Usage

```
DistanceProperties()
```

Details

The registry distinguishes the software convention of calling a quantity a “distance” from the mathematical definition of a metric. The properties are stated for the objects named in `ComparisonUnit`. For example, Euclidean and finite weighted Minkowski distances are metrics on represented points when all coordinate weights are positive, even when two labelled rows contain identical coordinates. Zero coordinate weights can produce a pseudometric, whereas `Tfidf_dist()` can map genuinely different feature rows to the same scalar weight and is therefore a pseudometric on the original rows.

The values “conditional” and “backend-dependent” indicate that the property cannot be asserted without the parameter or backend conditions in `Conditions`. Wrapper-specific native routes, including squared Euclidean and the correlation-based `*d` and `*m` families of `DistanceMatrix()`, have their own rows. External backend methods remain backend-dependent. The registry is documentation, not a runtime proof for an arbitrary user-supplied matrix.

Value

A data frame with one row per method or method family and columns:

Function Public `BIDistances` function or method family.

ComparisonUnit The mathematical objects compared.

NonNegative Whether non-negativity is guaranteed.

Symmetric Whether symmetry is guaranteed.

IdentityOfIndiscernibles Whether zero distance identifies the same comparison object.

TriangleInequality Whether the triangle inequality is guaranteed.

Classification Metric, pseudometric, divergence, dissimilarity, or a conditional combination of these terms.

Conditions Input and parameter conditions needed for the statement.

References

Deza MM, Deza E. Encyclopedia of Distances. Berlin, Heidelberg: Springer; 2009. doi:10.1007/978-3-642-00234-2.

See Also

[distance-conventions](#), [DistanceMatrix](#), [Minkowski_Distance](#)

Examples

```
properties = DistanceProperties()
properties[is.element(
  properties$Function,
  c(
    "Cosine_Distance", "Tfidf_dist", "EuclideanGPU_Distance",
    "Minkowski_Distance"
  )
), ]
```

DTW_Distance	<i>Dynamic Time Warping Distance</i>
--------------	--------------------------------------

Description

Dynamic Time Warping distance based on a Non-Euclidean pairwise distance.

Usage

```
DTW_Distance(Data, DistanceFunction, ...)
```

```
DTW_Distance(Data, DistanceFunction, ...)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
DistanceFunction	A function which calculates the pairwise distances in the same way as dist , which is the default when the argument is omitted.
...	Further arguments for DistanceFunction.

Details

Each column of Data is one time series of length n . The row order is the temporal order. No time variable or timestamp column should be given; this function does not require or accept a separate time vector.

Currently only implemented for time series of the same length. Distances in DistanceFunction should be calculated in the same way as in a two-dimensional data matrix, with the output of the function being an object of class [dist](#).

When this route is called through DistanceMatrix(), the wrapper method name occupies the argument name method. Use DistanceFunctionMethod there to forward a method name to DistanceFunction, for example DistanceFunctionMethod = "Manhattan".

Value

[1:n,1:n] numerical matrix of distances.

Author(s)

Michael Thrun

References

Giorgino, T. (2009). Computing and visualizing dynamic time warping alignments in R: the dtw package. *Journal of Statistical Software*, 31(7), 1–24.

See Also

[dtw](#), [DistanceMatrix](#), [MSMD_Distance](#), [TWED_Distance](#)

Examples

```
data(Hepta)
Data = Hepta$Data
if (requireNamespace("dtw", quietly = TRUE) &&
    requireNamespace("proxy", quietly = TRUE)) {
  Distance = DTW_Distance(
    Data,
    DistanceFunction = proxy::dist,
    method = "Manhattan"
  )
  Distance2 = DistanceMatrix(
    Data,
    method = "DTW_Distance",
    DistanceFunctionMethod = "Manhattan"
  )
}
```

EndresSchindelin_Distance

Endres–Schindelin Distance Matrix Between Variables

Description

Computes pairwise Endres–Schindelin distances between all columns of a numerical data matrix, using parallel processing when requested.

Usage

```
EndresSchindelin_Distance(Data, ncores = NULL)
```

```
EndresSchindelin_Distance(Data, ncores = NULL)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
ncores	NULL or one positive integer. With NULL, one less than the number of detected logical CPU cores is used, with a minimum of one. The number of workers never exceeds the number of variable pairs.

Details

For every pair of columns i, j , `endresSchindelin_TwoSamplePDF(Data[, i], Data[, j])` is evaluated. Results are inserted into both triangles of a full numerical matrix with zeros on the diagonal. A one-column input returns a one-by-one zero matrix.

The pairwise helper uses **ScatterDensity** to smooth both marginal histograms on a common grid. If **memshare** is installed, it is used to share the data matrix between PSOCK workers; otherwise a standard **parallel** fallback is used. Temporary clusters are stopped automatically. This function always returns an ordinary matrix and never a `dist` object.

Value

[1:n,1:n] numerical matrix of distances.

Author(s)

Michael Thrun

See Also

[endresSchindelin_TwoSamplePDF](#), [endresSchindelin_Distribution](#), [DistanceMatrix](#)

Examples

```
if (requireNamespace("ScatterDensity", quietly = TRUE)) {
  set.seed(123)
  Data = matrix(runif(100), nrow = 20, ncol = 5)
  Distance = EndresSchindelin_Distance(Data, ncores = 1)
  stopifnot(is.matrix(Distance))
}
```

`endresSchindelin_Distribution`

Endres–Schindelin Distance Between Probability Mass Vectors

Description

Computes the Endres–Schindelin metric between two aligned non-negative probability-mass vectors. The vectors are normalized internally.

Usage

```
endresSchindelin_Distribution(P, Q, eps = 1e-12)
```

Arguments

P	A non-empty finite numerical vector of non-negative masses.
Q	A finite numerical vector of the same length as P.
eps	One finite non-negative tolerance for tiny negative values caused by numerical rounding. Values between minus eps and zero are set to zero; more negative values are rejected. This argument is not additive smoothing.

Details

Each input is divided by its own positive sum. For the resulting probability vectors, the distance is

$$D(P, Q) = \sqrt{\sum_i \left[p_i \log_2 \left(\frac{2p_i}{p_i + q_i} \right) + q_i \log_2 \left(\frac{2q_i}{p_i + q_i} \right) \right]}.$$

Zero-probability terms contribute zero. This equals the square root of twice the Jensen–Shannon divergence when logarithms use base two.

Value

One non-negative numerical value.

Author(s)

Michael Thrun

References

Endres, D. M., and Schindelin, J. E. (2003). A new metric for probability distributions. IEEE Transactions on Information Theory, 49(7), 1858–1860.

Examples

```
P = c(0.2, 0.5, 0.3)
Q = c(0.1, 0.7, 0.2)
endresSchindelin_Distribution(P, Q)
```

endresSchindelin_TwoSamplePDF

Endres–Schindelin Distance Between Two Samples

Description

Estimates the Endres–Schindelin distance between two univariate samples after estimating both marginal densities on one common numerical support grid.

Usage

```
endresSchindelin_TwoSamplePDF(x, y, grid.length = 512L, lambda = 4)
```

Arguments

<code>x</code>	A finite numerical vector with at least two observations from the first distribution.
<code>y</code>	A finite numerical vector with at least two observations from the second distribution. Its length need not equal the length of <code>x</code> .
<code>grid.length</code>	One integer greater than or equal to 16, giving the number of points on the common support grid.
<code>lambda</code>	One positive finite smoothing factor passed unchanged to <code>ScatterDensity::SmoothedDensitiesXY()</code> . The historical default is 4.

Details

A radius is estimated separately for each sample with `DataVisualizations::ParetoRadius()`. If a positive radius cannot be obtained, as can occur for a constant sample, a positive numerical fall-back is used to extend the pooled range. Both samples are binned and smoothed independently with `ScatterDensity::SmoothedDensitiesXY()`, using the same support grid and the same `lambda`. Consequently, entry i of each normalized marginal density represents the same coordinate. The two samples do not need to be paired and may have different lengths.

For probability vectors $P = (p_i)$ and $Q = (q_i)$, the function returns

$$D(P, Q) = \sqrt{\sum_i \left[p_i \log_2 \left(\frac{2p_i}{p_i + q_i} \right) + q_i \log_2 \left(\frac{2q_i}{p_i + q_i} \right) \right]}.$$

Zero-probability terms contribute zero. Constant and unequal-length samples are supported.

Value

One non-negative numerical value.

Author(s)

Michael Thrun

References

Endres, D. M., and Schindelin, J. E. (2003). A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7), 1858–1860.

See Also

[endresSchindelin_Distribution](#), [EndresSchindelin_Distance](#)

Examples

```
if (requireNamespace("ScatterDensity", quietly = TRUE)) {
  set.seed(123)
  x = rnorm(500)
  y = rnorm(300, mean = 1)
```

```

endresSchindelin_TwoSamplePDF(x, y)
stopifnot(endresSchindelin_TwoSamplePDF(rep(2, 20), rep(2, 30)) < 1e-10)
}

```

EuclideanGPU_Distance *Weighted Euclidean Distances with an Optional OpenCL Backend*

Description

Computes all pairwise weighted Euclidean distances between rows of a numerical matrix. An optional **OpenCL** backend can use an available accelerator; a CPU implementation is always available.

Usage

```

EuclideanGPU_Distance(
  Data,
  Weights,
  Mem = 2,
  OutputType = "mat",
  ctx = NULL,
  backend = c("auto", "opencl", "cpu"),
  threads = 2L
)

```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
Weights	Optional non-negative numerical vector with one value per column of Data. The default is a vector of ones.
Mem	One finite positive number giving the device-memory budget in GiB. The OpenCL planner uses a conservative estimate of eight bytes per value.
OutputType	One of "mat", "dist", or "vec". "mat" returns the full square matrix, "dist" returns a dist object, and "vec" returns the full matrix as a column-major vector of length nrow(Data)^2.
ctx	An optional OpenCL context created by <code>OpenCL::oclContext()</code> . It is ignored by the CPU backend. When it is NULL, the OpenCL backend creates a context with precision = "best".
backend	Backend selection. "auto" tries OpenCL and falls back to the CPU with a warning when the package, platform, device, context, kernel, or allocation is unavailable. "opencl" requires OpenCL and reports an error instead of falling back. "cpu" never probes OpenCL.
threads	Number of CPU threads used by the CPU backend or an automatic fallback. Must be either 1 or 2; the default is 2 to respect CRAN's shared-resource policy. It is ignored after a successful OpenCL calculation.

Details

For rows x_i and x_j , the returned value is

$$\left(\sum_k w_k (x_{ik} - x_{jk})^2 \right)^{1/2}.$$

The **OpenCL** package is suggested rather than imported, so BIDsances can be installed and checked on systems without OpenCL headers, drivers, or hardware. Depending on the memory budget, the OpenCL path uses a full-matrix, output-batched, or input-blocked implementation.

With precision = "best", OpenCL may use single precision on devices without double-precision support. Small numerical differences from the CPU result are therefore possible.

Value

[1:n,1:n] numerical matrix of distances. With OutputType = "dist", the result is returned as a dist object; with OutputType = "vec", the full matrix is returned as a column-major numerical vector.

Author(s)

Quirin Stier, Michael Thrun, Luca Brinkmann

See Also

[calculateMemoryDemandGPU](#), [DistanceMatrix](#), [dist](#)

Examples

```
Data = as.matrix(iris[1:8, 1:4])
Weights = c(1, 2, 0.5, 1)

D = EuclideanGPU_Distance(
  Data,
  Weights,
  backend = "cpu",
  OutputType = "mat"
)
reference = as.matrix(stats::dist(sweep(Data, 2, sqrt(Weights), "*")))
dimnames(reference) = dimnames(D)
stopifnot(isTRUE(all.equal(D, reference)))

## Not run:
## This block requires an installed OpenCL implementation and usable device.
Dgpu = EuclideanGPU_Distance(
  Data,
  Weights,
  backend = "opencl",
  OutputType = "mat"
)
stopifnot(isTRUE(all.equal(Dgpu, reference, tolerance = 1e-5)))

## End(Not run)
```

EuclideanMulticore_Distance

Fast Pairwise Euclidean Distance Matrix

Description

Computes Euclidean distances between all rows of a numerical matrix.

Usage

```
EuclideanMulticore_Distance(X)
```

Arguments

X A non-empty finite numerical matrix with observations in rows.

Value

[1:d,1:d] numerical matrix of distances.

Author(s)

Michael Thrun

Examples

```
X = matrix(c(0, 0, 1, 0, 0, 2), ncol = 2, byrow = TRUE)
D = EuclideanMulticore_Distance(X)
stopifnot(is.matrix(D))
```

fast_dip_stat

Fast Hartigan Dip Statistic

Description

Computes Hartigan's dip statistic with the compiled corrected AS 217 algorithm. Unsorted input is copied and sorted once in C++; a pre-sorted double vector can be processed without that data copy.

Usage

```
fast_dip_stat(
  x,
  sorted = FALSE,
  check.sorted = TRUE,
  min.is.0 = FALSE
)
```

Arguments

<code>x</code>	Numeric vector. Missing values and NaNs are removed before calculation.
<code>sorted</code>	Logical scalar indicating whether <code>x</code> is already sorted in nondecreasing order. Default is FALSE.
<code>check.sorted</code>	Logical scalar. With <code>sorted = TRUE</code> , verify that the complete values are in non-decreasing order. Set this to FALSE only when ordering is guaranteed. Default is TRUE.
<code>min.is.0</code>	Logical scalar. Use zero instead of $1/(2*n)$ as the minimum dip. The default FALSE follows <code>diptest::dip()</code> .

Details

After sorting, the dip kernel is linear in the sample size. For unsorted data, total complexity is dominated by sorting and is therefore $O(n \log n)$. The corrected AS 217 algorithm constructs the greatest convex minorant and least concave majorant with four internal index arrays.

For a pre-sorted vector of storage mode double, `sorted = TRUE` avoids the ordinary value-vector copy. Disabling `check.sorted` also avoids the linear ordering check, but incorrect ordering then produces an invalid statistical result.

Value

A numerical scalar containing the dip statistic. Attribute `n` gives the number of complete observations, and attribute `modal.interval` gives the one-based lower and upper indices of the modal interval in the sorted sample.

Author(s)

Hartigan dip algorithm; C++ and R interface integrated into **BIDistances** by OpenAI.

References

Hartigan, J. A. and Hartigan, P. M. (1985). The Dip Test of Unimodality. *The Annals of Statistics*, **13**(1), 70–84.

Maechler, M. and contributors. **diptest**: Hartigan’s Dip Test Statistic for Unimodality.

See Also

[fast_dip_test](#), `diptest::dip`

Examples

```
fast_dip_stat(c(1, 1, 2, 2))

set.seed(1)
xs <- sort(rnorm(1000))
fast_dip_stat(xs, sorted = TRUE)
```

fast_dip_test

*Fast Hartigan Dip Test for Large Samples***Description**

Performs Hartigan's dip test with a compiled dip kernel, the finite-sample critical-value table used by **diptest**, an explicit large-sample calibration, and an optional simulation path based on exact uniform order statistics.

Usage

```
fast_dip_test(
  x,
  simulate.p.value = FALSE,
  B = 2000,
  p.value.method = c("auto", "table", "asymptotic", "bootstrap"),
  sorted = FALSE,
  check.sorted = TRUE,
  min.is.0 = FALSE,
  warn.asymptotic = FALSE
)

dip.test.fast(
  x,
  simulate.p.value = FALSE,
  B = 2000,
  p.value.method = c("auto", "table", "asymptotic", "bootstrap"),
  sorted = FALSE,
  check.sorted = TRUE,
  min.is.0 = FALSE,
  warn.asymptotic = FALSE
)

dip_test_fast(
  x,
  simulate.p.value = FALSE,
  B = 2000,
  p.value.method = c("auto", "table", "asymptotic", "bootstrap"),
  sorted = FALSE,
  check.sorted = TRUE,
  min.is.0 = FALSE,
  warn.asymptotic = FALSE
)
```

Arguments

x Numeric vector. Missing values and NaNs are removed before calculation.

simulate.p.value	Logical scalar. Use Monte-Carlo calibration with exact uniform order-statistic draws. This overrides p.value.method and selects "bootstrap".
B	Positive integer number of Monte-Carlo replicates when simulation is used. Default is 2000.
p.value.method	Calibration method. "auto" uses finite-sample interpolation through $n = 72000$ and asymptotic calibration above 72000. "table" refuses larger samples. "asymptotic" always uses the last scaled critical-value row. "bootstrap" uses simulation.
sorted	Logical scalar indicating whether x is already sorted. For a double vector, this enables the no-copy input path.
check.sorted	Logical scalar. Validate ordering when sorted = TRUE.
min.is.0	Logical scalar. Use zero instead of $1/(2*n)$ as the minimum dip.
warn.asymptotic	Logical scalar. Warn when "auto" switches to asymptotic calibration above 72000. Default is FALSE.

Details

The table path interpolates scaled quantiles of $\sqrt{n}D_n$. Above the largest tabulated sample size, 72000, the last scaled row is treated as an approximation to the limiting distribution. This is also the large-sample rule used by `dip.test::dip.test()`, but here it is exposed explicitly through `p.value.method`.

The dip statistic itself is computed from all complete observations. The large-sample path does not downsample the data.

The bootstrap generates exact uniform order statistics using normalized exponential spacings, so each replicate does not need to be sorted. It still requires $O(Bn)$ work and is usually unsuitable when both n and B are large.

`dip.test.fast` and `dip.test_fast` are aliases of `fast_dip_test`.

Value

An object of class "htest". Alongside the standard test fields, the object contains:

nobs	Number of complete observations.
p.value.method	Calibration actually used.
table.max.n	Largest finite-sample table size, 72000.
modal.interval	One-based modal-interval indices in the sorted sample.

Author(s)

Hartigan dip algorithm; C++ and R interface integrated into **BIDistances** by OpenAI.

References

- Hartigan, J. A. and Hartigan, P. M. (1985). The Dip Test of Unimodality. *The Annals of Statistics*, **13**(1), 70–84.
- Maechler, M. and contributors. **dip.test**: Hartigan's Dip Test Statistic for Unimodality.

See Also

[fast_dip_stat](#), [fast_dip_test_matrix](#), [diptest::dip.test](#)

Examples

```
set.seed(1)
x <- c(rnorm(5000, -2), rnorm(5000, 2))
fast_dip_test(x)

xs <- sort(x)
fast_dip_test(xs, sorted = TRUE, check.sorted = FALSE)
```

`fast_dip_test_matrix` *Fast Hartigan Dip Tests for Matrix Columns*

Description

Computes Hartigan dip statistics and calibrated p-values for several distributions stored in matrix columns. All selected columns are processed in one native call with a reusable linear workspace.

Usage

```
fast_dip_test_matrix(
  x,
  columns = NULL,
  p.value.method = c("auto", "table", "asymptotic"),
  min.is.0 = FALSE
)
```

Arguments

<code>x</code>	Numeric matrix or numeric data frame. Every selected column is treated as one independent distribution.
<code>columns</code>	Columns to test. Use <code>NULL</code> for all columns, or supply positive whole column indices, a logical selector with one value per column, or character column names. Duplicate selections are removed while preserving order.
<code>p.value.method</code>	Calibration method. "auto" uses finite-sample interpolation through 72000 and asymptotic calibration above 72000. "table" rejects a selected column with more than 72000 complete values. "asymptotic" always uses the last scaled table row.
<code>min.is.0</code>	Logical scalar. Use zero instead of $1/(2*n)$ as the minimum dip.

Details

The native implementation reads the original matrix without first creating a selected-column sub-matrix. It copies and sorts one selected column at a time and reuses the same dip workspace for the next column. Thus the work arrays do not scale with the number of selected distance distributions.

Missing values and NaNs are removed independently from every column. A column with no complete observations causes an error.

Bootstrap calibration is deliberately not available in this batch function, because its $O(Bn)$ cost applies independently to every column. Use `fast_dip_test()` for a simulated p-value of a single vector.

`DistanceDistributionAnalysis()` uses this function automatically for all analyzable distance distributions when the number of pairwise values is strictly greater than 72000.

Value

A data frame with one row per selected column and variables:

Distance	Column name, or a generated name when none is available.
Column	One-based source column index.
N	Number of complete values in the column.
DipStatistic	Hartigan dip statistic.
DipPValue	Calibrated p-value.
PValueMethod	Calibration actually used.
ModalIntervalLow, ModalIntervalHigh	One-based modal-interval indices in the sorted complete sample.

Author(s)

OpenAI; integrated into **BIDistances** for efficient multi-distance analysis.

See Also

[fast_dip_test](#), [fast_dip_stat](#), [DistanceDistributionAnalysis](#)

Examples

```
set.seed(1)
X <- cbind(
  Unimodal = rnorm(5000),
  Bimodal = c(rnorm(2500, -2), rnorm(2500, 2))
)
fast_dip_test_matrix(X)
```

Fractional_Distance *Fractional Minkowski Pairwise Distances*

Description

Computes $(\sum_k |x_{ik} - x_{jk}|^p)^{1/p}$ between all pairs of cases.

Usage

```
Fractional_Distance(Data, p)
```

```
Fractional_Distance(Data, p)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
p	One positive finite exponent. Values below one are allowed, although the resulting dissimilarity is not a metric.

Details

For $p \geq 1$, the function computes the Minkowski metric. For $0 < p < 1$, the same formula defines a fractional dissimilarity that generally does not satisfy the triangle inequality.

Value

[1:n,1:n] numerical matrix of distances.

Author(s)

Michael Thrun

References

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space, Database Theory, ICDT 2001, pp. 420-434, DOI: 10.1007/3-540-44503-X_27.

See Also

[DistanceMatrix](#)

Examples

```
data(Hepta)
Distance = Fractional_Distance(Hepta$Data, p = 1/2)
stopifnot(is.matrix(Distance))
```

Gini_Distance

*Gini Distance Between Cases***Description**

Computes a full pairwise matrix between the cases of a numerical data matrix, based on absolute differences between their Gini coefficients.

Usage

```
Gini_Distance(Data)
```

```
Gini_Distance(Data)
```

Arguments

Data [1:n,1:d] numerical matrix with n cases, d variables

Details

The Gini coefficient G_i is calculated for every original case i using `ineq::ineq(..., type = "Gini")`. The returned entry is

$$D_{ij} = |G_i - G_j|.$$

The implementation operates on the rows of Data.

Value

[1:n,1:n] numerical matrix of distances.

Author(s)

Michael Thrun

See Also

[ineq](#), [DistanceMatrix](#)

Examples

```
if (requireNamespace("ineq", quietly = TRUE)) {
  set.seed(123)
  Data = matrix(runif(50), nrow = 10, ncol = 5)
  Distance = Gini_Distance(Data)
  stopifnot(is.matrix(Distance))
}
```

Hearingloss_N109	<i>Hearingloss data</i>
------------------	-------------------------

Description

Hearingloss data, with Gene2GoTerm matrix.

Usage

```
data('Hearingloss_N109')
```

Details

FeatureMarix_Gene2Term contains the dataset, NCBI are the row names for the genes and GoTerm_Header contains the column names for the GoTerms. Size of data matrix is 109 with dimension 829.

Source

[NCBI OtoGenome Test for Hearing Loss](#), accessed 24 June 2022.

References

GeneTestingRegistry (2018). OtoGenome Test for Hearing Loss Retrieved 2017

Examples

```
data(Hearingloss_N109)
str(Hearingloss_N109)
```

Hellinger_Distance	<i>Hellinger Distance Between Binary-Class Conditional Densities</i>
--------------------	--

Description

For every feature, estimates its conditional density in classes 1 and 2 on a common domain and computes their Hellinger distance.

Usage

```
Hellinger_Distance(Data, Cls)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
Cls	A finite numerical vector with one class label per row, containing exactly the values 1 and 2. Each class must have at least two observations.

Value

A numerical matrix with one row per feature and columns Feature and HellingerDistance.

Author(s)

Quirin Stier

Examples

```
Data = as.matrix(iris[1:100, 1:4])
Cls = as.numeric(iris[1:100, 5])
Hellinger_Distance(Data, Cls)
```

Hepta

Hepta introduced in [Utsch, 2003]

Description

Clearly defined clusters, different variances. Detailed description of dataset and its clustering challenge is provided in [Thrun/Utsch, 2020].

Usage

```
data('Hepta')
```

Details

Size 212, Dimensions 3, stored in Hepta\$Data

Classes 7, stored in Hepta\$Cls

References

[Utsch, 2003] Utsch, A.: Maps for the visualization of high-dimensional data spaces, Proc. Workshop on Self organizing Maps (WSOM), pp. 225-230, Kyushu, Japan, 2003.

[Thrun/Utsch, 2020] Thrun, M. C., & Utsch, A.: Clustering Benchmark Datasets Exploiting the Fundamental Clustering Problems, Data in Brief, Vol. 30(C), pp. 105501, doi:[10.1016/j.dib.2020.105501](https://doi.org/10.1016/j.dib.2020.105501), 2020.

Examples

```
data(Hepta)
str(Hepta)
```

IntraClusterBoundaryCounts

Count Intra-Cluster Distances Above a Boundary

Description

Computes, for every cluster, the absolute number and empirical proportion of unique intra-cluster pairwise distances that exceed a specified Bayesian boundary.

For each cluster, only the upper triangle of the corresponding cluster-specific distance matrix is used, so every unordered pair of observations contributes exactly once.

Usage

```
IntraClusterBoundaryCounts(
  Distance,
  CIs,
  Boundary
)
```

Arguments

Distance	Numeric square distance matrix of dimension $n \times n$. Rows and columns correspond to the observations described by CIs.
CIs	Numeric or character vector of length n containing the cluster label for every observation. Missing cluster labels are not allowed.
Boundary	Finite numeric scalar defining the Bayesian boundary. Distances strictly larger than this value are counted as exceedances.

Details

Let C_k denote cluster k with n_k observations. The number of unique intra-cluster distances is

$$N_k = \frac{n_k(n_k - 1)}{2}.$$

The function extracts the upper triangle of the cluster-specific distance matrix, excluding the diagonal. Consequently, every unordered pair contributes exactly once.

The exceedance count is

$$E_k = \#\{z : z > b\},$$

where b is Boundary. The empirical exceedance proportion is

$$v_k = \frac{E_k}{N_k}.$$

Distances exactly equal to the boundary are counted in AtOrBelowBoundary.

Singleton clusters have no intra-cluster pairs. For these clusters, IntrapartitionPairs_Nk, AtOrBelowBoundary, and AboveBoundary_Ek are zero, while ExceedanceProportion_vk is NA_real_.

All intra-cluster distances must be finite. Non-finite distances are rejected because the denominator N_k assumes that every unique intra-cluster pair enters the calculation.

Cluster labels are processed in sorted unique order.

Value

A data frame with one row per cluster and the following columns:

Cluster	Cluster label.
Cases_nk	Number of observations n_k assigned to the cluster.
IntrapartitionPairs_Nk	Number of unique intra-cluster observation pairs, $N_k = n_k(n_k - 1)/2$.
AtOrBelowBoundary	Number of unique intra-cluster distances less than or equal to Boundary.
AboveBoundary_Ek	Number of unique intra-cluster distances strictly greater than Boundary.
ExceedanceProportion_vk	Empirical exceedance proportion $v_k = E_k/N_k$. For singleton clusters, where $N_k = 0$, this value is NA_real_.
Boundary	Copy of the supplied boundary value for each cluster.

Note

The function checks that Distance is numeric and square, that `length(Cls) == nrow(Distance)`, that cluster labels are complete, and that Boundary is finite.

It does not independently verify symmetry, non-negativity, or a zero diagonal. Those properties should already hold for a valid distance matrix.

Author(s)

Michael Thrun

References

Thrun, M. C. (2021). The Exploitation of Distance Distributions for Clustering. *International Journal of Computational Intelligence and Applications*, **20**(3), 2150016. doi: 10.1142/S1469026821500164.

Examples

```
Distance <- matrix(
  c(
    0, 1, 4, 5,
    1, 0, 3, 6,
    4, 3, 0, 2,
    5, 6, 2, 0
  ),
```

```

    nrow = 4,
    byrow = TRUE
  )

  Cls <- c("A", "A", "B", "B")

  IntraClusterBoundaryCounts(
    Distance = Distance,
    Cls = Cls,
    Boundary = 1.5
  )

```

Jaccard_Distance	<i>Generalized Jaccard Distance Between Cases</i>
------------------	---

Description

Computes generalized (weighted) Jaccard distances between rows of a finite non-negative data matrix.

Usage

```
Jaccard_Distance(Data)
```

Arguments

Data [1:n,1:d] numerical matrix with n cases, d variables

Details

For two rows x and y , the distance is

$$1 - \frac{\sum_k \min(x_k, y_k)}{\sum_k \max(x_k, y_k)}.$$

This is equivalent to the quantitative Jaccard form derived from Bray–Curtis dissimilarity. Two all-zero rows have zero union mass and are assigned distance zero.

Value

An $n \times n$ symmetric numerical distance matrix with zeros on the diagonal.

Author(s)

Michael Thrun

See Also[DistanceMatrix](#)**Examples**

```
Data = rbind(c(1, 0, 1), c(1, 1, 0), c(0, 0, 1))
Distance = Jaccard_Distance(Data)
Jaccard_Distance(rbind(c(0, 0), c(0, 0)))
```

KNearestNeighborsFromDistance

Find Nearest Neighbors from a Distance Matrix

Description

For every point, finds the k nearest other points represented by a square distance matrix. The point itself is excluded explicitly rather than by assuming that the diagonal is the unique smallest value.

Usage

```
KNearestNeighborsFromDistance(k, Distances)
```

Arguments

<code>k</code>	A single integer giving the number of nearest neighbors to return. It must satisfy $1 \leq k < n$, where n is the number of points.
<code>Distances</code>	A square numerical distance matrix, a <code>dist</code> object, or a numerical condensed-distance vector accepted by <code>pracma::squareform()</code> . Row i represents point i .

Details

Missing values are not accepted. Infinite distances are allowed. Equal finite or infinite distances are ordered by the point index, with the smaller index first.

Value

A list with two components:

<code>NNind</code>	An integer matrix with n rows and k columns. Row i contains the indices of the nearest neighbors of point i .
<code>NNdists</code>	A numerical matrix with n rows and k columns. Row i contains the corresponding distances.

Author(s)

Michael Thrun

Examples

```
Distances = matrix(
  c(
    0, 1, 4, 1,
    1, 0, 2, 3,
    4, 2, 0, 5,
    1, 3, 5, 0
  ),
  nrow = 4,
  byrow = TRUE
)

KNearestNeighborsFromDistance(2, Distances)
```

Kullback_Leibler_div *Smoothed Kullback–Leibler Divergence*

Description

Computes a directed or symmetric Kullback–Leibler divergence from two empirical discrete samples or two aligned probability-mass vectors.

Usage

```
Kullback_Leibler_div(P, Q, sym = 1, PDF = FALSE, Eps = 1/10000)
```

Arguments

P	When PDF = FALSE, an atomic vector containing the first sample. When PDF = TRUE, a numerical vector of non-negative probability masses on the same ordered bins as Q.
Q	The second sample or aligned probability-mass vector.
sym	TRUE or 1 returns the sum of both directed divergences. FALSE or 0 returns only the divergence from P to Q.
PDF	Logical. If FALSE, empirical category counts are constructed on the union of the observed categories. If TRUE, P and Q are interpreted as aligned probability masses.
Eps	One positive finite additive smoothing value. The value is added to every support bin before normalization. The default is 1/10000.

Details

For empirical samples, non-finite observations are removed independently from P and Q. Let x be their joint discrete support and let $K = |x|$, named `AnzUniqX` in the implementation. The count in every one of the K bins receives the pseudocount `Eps`, after which each count vector is normalized to sum to one. For PDF = TRUE, aligned non-finite bins are removed, `Eps` is added to every remaining bin, and the two vectors are normalized separately.

The package's category-normalized directed divergence is

$$D(P\|Q) = \frac{1}{K} \sum_{i=1}^K p_i \log \left(\frac{p_i}{q_i} \right).$$

Thus, the result is deliberately divided by `AnzUniqX`. With `sym = 1`, the returned value is $D(P\|Q) + D(Q\|P)$, with each directed term divided by the same number of bins. Natural logarithms are used.

Value

A list with components:

KLD	The category-normalized directed or symmetric divergence.
p	The smoothed, normalized probabilities for P.
q	The smoothed, normalized probabilities for Q.
x	The sorted joint support for empirical samples, or NULL when PDF = TRUE.

Author(s)

Michael Thrun

Examples

```
result = Kullback_Leibler_div(c(0, 0, 1), c(1, 1, 1), sym = FALSE)
result$KLD
stopifnot(all(result$p > 0), all(result$q > 0))

pdf_result = Kullback_Leibler_div(
  c(0.5, 0.5, 0), c(0.2, 0.3, 0.5), PDF = TRUE
)
stopifnot(abs(sum(pdf_result$p) - 1) < 1e-12)
```

Mahalanobis_Distance *Pairwise Squared Generalized Mahalanobis Distances*

Description

Computes squared generalized Mahalanobis distances between all pairs of rows of a numerical matrix.

Usage

```
Mahalanobis_Distance(X, cov, inverted = FALSE)
```

Arguments

<code>x</code>	a matrix of data ($n \times d$) n cases, d variables
<code>cov</code>	A finite symmetric positive-definite $d \times d$ covariance matrix, or a symmetric positive-definite precision matrix when <code>inverted = TRUE</code> .
<code>inverted</code>	Logical indicating whether <code>cov</code> is already the inverse covariance matrix.

Details

The returned entry is

$$D_{ij}^2 = (x_i - x_j)^T \Sigma^{-1} (x_i - x_j).$$

Thus the function returns squared distances rather than their square roots. A Cholesky factorization verifies positive definiteness; singular or indefinite matrices are rejected.

Value

An $n \times n$ symmetric numerical matrix of squared distances.

Author(s)

Anderson Rodrigo da Silva

References

Mahalanobis, P. C. (1936). On the generalized distance in statistics. Proceedings of the National Institute of Sciences of India.

Examples

```
# Manly (2004, p.65-66)
x1 = c(131.37, 132.37, 134.47, 135.50, 136.17)
x2 = c(133.60, 132.70, 133.80, 132.30, 130.33)
x3 = c(99.17, 99.07, 96.03, 94.53, 93.50)
x4 = c(50.53, 50.23, 50.57, 51.97, 51.37)
x = cbind(x1, x2, x3, x4)
Cov = matrix(c(21.112,0.038,0.078,2.01, 0.038,23.486,5.2,2.844,
  0.078,5.2,24.18,1.134, 2.01,2.844,1.134,10.154), 4, 4)
Mahalanobis_Distance(x, Cov)

# End (not run)
```

Minkowski_Distance	<i>Weighted Minkowski Distances with Selectable CPU and OpenCL Backends</i>
--------------------	---

Description

Computes all pairwise weighted Minkowski distances between rows of a numerical matrix through **parallelDist**, an internal **RcppParallel** multicore implementation, or an optional **OpenCL** backend.

Usage

```
Minkowski_Distance(
  Data,
  p = 2,
  Weights = NULL,
  backend = "auto",
  Mem = 2,
  OutputType = "mat",
  ctx = NULL,
  threads = 2L
)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
p	One numerical Minkowski exponent greater than or equal to 1, or Inf. The default is 2. Values below 1 are not metrics and should be computed with <code>Fractional_Distance()</code> when required.
Weights	Optional non-negative numerical vector with one value per column of Data. The default is a vector of ones. For finite p, the weights enter inside the sum. For p = Inf, only unit weights are accepted.
backend	One of "auto", "parallelDist", "multicore", or "opencl". Matching is case-insensitive. "auto" tries OpenCL and falls back to the internal multicore backend with a warning. "opencl" requires a usable OpenCL installation and does not fall back.
Mem	One finite positive number giving the device-memory budget in GiB. It is used by "auto" and "opencl".
OutputType	One of "mat", "dist", or "vec". "mat" returns the full square matrix, "dist" returns a dist object, and "vec" returns the full matrix as a column-major vector of length <code>nrow(Data)^2</code> .
ctx	An optional OpenCL context created by <code>OpenCL::oclContext()</code> . When it is NULL, the OpenCL backend creates a context with <code>precision = "best"</code> . It is ignored by the CPU backends.
threads	One positive integer giving the number of CPU threads for the "parallelDist" or "multicore" backend and for the multicore fallback from "auto".

Details

For rows x_i and x_j , finite p uses

$$d_{p,w}(x_i, x_j) = \left(\sum_k w_k |x_{ik} - x_{jk}|^p \right)^{1/p}.$$

The implementation scales coordinate k by $w_k^{1/p}$. Strictly positive weights give a metric for $p \geq 1$. One or more zero weights can map distinct rows to the same represented point, yielding a pseudometric. Negative weights are rejected.

The special cases are weighted Manhattan distance for $p = 1$, weighted Euclidean distance for $p = 2$, and the maximum or Chebyshev distance for $p = \text{Inf}$. Because positive finite weights disappear in the mathematical limit as p tends to infinity, this interface accepts $p = \text{Inf}$ only with unit weights.

The "parallelDist" backend scales the columns and calls `parallelDist::parDist()`. The "multicore" backend uses an internal **RcppParallel** worker and applies a scaled computation for general finite p to reduce overflow and underflow in intermediate powers.

For backend = "opencl" and $p = 2$, the function delegates to the established `EuclideanGPU_Distance()` implementation. Its existing GPU R code and three Euclidean kernels remain unchanged. Other values of p use separate Minkowski OpenCL kernels for complete, output-batched, and input-blocked execution according to the available memory budget.

With an OpenCL context using single precision, small numerical differences from CPU results are possible. Use backend = "parallelDist" or backend = "multicore" when OpenCL is not required.

Value

[1:n,1:n] numerical matrix of distances. With `OutputType = "dist"`, the result is returned as a `dist` object; with `OutputType = "vec"`, the full matrix is returned as a column-major numerical vector.

See Also

[EuclideanGPU_Distance](#), [EuclideanMulticore_Distance](#), [Fractional_Distance](#), [DistanceMatrix](#), [calculateMemoryDemandGPU](#)

Examples

```
Data = as.matrix(iris[1:8, 1:4])
Weights = c(1, 2, 0.5, 1)

Dparallel = Minkowski_Distance(
  Data,
  p = 3,
  Weights = Weights,
  backend = "parallelDist",
  threads = 2
)
Dmulticore = Minkowski_Distance(
  Data,
  p = 3,
```

```

    Weights = Weights,
    backend = "multicore",
    threads = 2
)
stopifnot(isTRUE(all.equal(Dparallel, Dmulticore, tolerance = 1e-10)))

Deuclidean = Minkowski_Distance(
  Data,
  p = 2,
  Weights = Weights,
  backend = "multicore"
)
reference = as.matrix(stats::dist(sweep(Data, 2, sqrt(Weights), "*")))
dimnames(reference) = dimnames(Deuclidean)
stopifnot(isTRUE(all.equal(Deuclidean, reference)))

## Not run:
## This block requires an installed OpenCL implementation and usable device.
Dopencl = Minkowski_Distance(
  Data,
  p = 3,
  Weights = Weights,
  backend = "opencl"
)
stopifnot(isTRUE(all.equal(Dopencl, Dparallel, tolerance = 1e-5)))

## End(Not run)

```

MSMD_Distance

Move-Split-Merge Distance

Description

Computes the Move-Split-Merge distance between two univariate time series.

Usage

```
MSMD_Distance(Values1, Values2, ParameterC)
```

```
MSMD_Distance(Values1, Values2, ParameterC)
```

Arguments

Values1	A non-empty finite numerical vector containing the first time series. The entry order is the temporal order.
Values2	A non-empty finite numerical vector containing the second time series. The entry order is the temporal order.
ParameterC	One positive finite scalar giving the base cost of every split or merge operation.

Details

The dynamic program permits move, split, and merge operations. A move costs the absolute difference between matched values. A split or merge always costs at least `ParameterC`; when the new value lies outside the interval defined by the preceding and opposing values, the smaller endpoint deviation is added. Changing `ParameterC` therefore changes the relative cost of changing series length.

No time variable is required or accepted. The sequence order is defined by the order of the vector entries. In `DistanceMatrix()`, each column of `Data` is one time series and the row order is the temporal order. The matrix conventions are documented in [DistanceMatrix](#).

Value

One non-negative numerical distance.

Author(s)

Quirin Stier

References

Holznigenkemper, J., C. Komusiewicz, and B. Seeger, On computing exact means of time series using the move-split-merge metric, *Data Mining and Knowledge Discovery*, 2023. 37(2): p. 595-626

See Also

[DistanceMatrix](#), [DTW_Distance](#)

Examples

```
MSMD_Distance(c(1, 2, 3), c(1, 2, 4), ParameterC = 1)
stopifnot(MSMD_Distance(1:4, 1:4, ParameterC = 1) == 0)

Data = cbind(Series1 = 1:4, Series2 = c(1, 2, 3, 5))
Distance = DistanceMatrix(Data, "MSMD_Distance", ParameterC = 1)
```

NearestNeighborIndex *Index of the Nearest Other Row*

Description

Returns the row index of the nearest other observation to a specified row.

Usage

```
NearestNeighborIndex(Data, i, defined)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
i	One valid row index of Data.
defined	Optional logical or 0/1 vector selecting columns. All columns are used when it is omitted.

Details

The query row is excluded by its index rather than by assuming that the first zero distance is the query itself. This remains correct when duplicate rows are present. Equal-distance ties are resolved by the smaller row index.

Value

One integer row index.

Author(s)

Michael Thrun and Raphael Paebst

Examples

```
X = rbind(c(0, 0), c(0, 0), c(2, 2))
stopifnot(NearestNeighborIndex(X, 1) == 2)
```

SharedNeighbor_Distance

Shared-Nearest-Neighbor Distance

Description

Computes the dissimilarity $1 - s_{ij}/k$, where s_{ij} is the number of neighbors shared by cases i and j among their k nearest Euclidean neighbors.

Usage

```
SharedNeighbor_Distance(
  Data, k = 5, NThreads = NULL, ComputationInR = FALSE,
  verbose = FALSE
)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
k	Integer defining the number of nearest neighbors.
NThreads	NULL or one positive integer controlling both the Euclidean-distance backend and neighbor-list parallelization.
ComputationInR	Logical. If FALSE, the shared-neighbor matrix is computed by the compiled implementation; if TRUE, a reference R loop is used.
verbose	Logical. If TRUE, report elapsed computation time.

Details

The query case itself is excluded explicitly from its neighbor list, including when duplicate cases create several zero Euclidean distances. Any temporary PSOCK cluster is stopped automatically, including after an error.

Value

[1:n,1:n] numerical matrix of distances.

Author(s)

Quirin Stier

See Also

[DistanceMatrix](#)

Examples

```
Data = rbind(c(0, 0), c(0, 1), c(1, 0), c(4, 4))
Distance = SharedNeighbor_Distance(
  Data, k = 2, NThreads = 1, ComputationInR = TRUE
)
stopifnot(is.matrix(Distance), isTRUE(all.equal(Distance, t(Distance))))

data(Hepta)
Distance = SharedNeighbor_Distance(
  Hepta$Data, NThreads = 1, ComputationInR = TRUE
)
```

Tfidf_Distance	<i>TF-IDF-Derived Distance Between Genes</i>
----------------	--

Description

Computes the term-frequency-inverse-document-frequency (TF-IDF) derived distance for a gene-by-GO-term feature matrix. GO terms are interpreted as documents and genes as terms.

Usage

```
Tfidf_Distance(FeatureMatrix_Gene2GoTerm, tf_fun = mean)
```

```
Tfidf_dist(FeatureMatrix_Gene2GoTerm, tf_fun = mean)
```

Arguments

FeatureMatrix_Gene2GoTerm	A non-empty, finite, non-negative numeric matrix with n genes in rows and m GO terms in columns. Entry x_{ij} is the number of annotations of gene i to GO term j ; values greater than one can represent multiple evidence codes. Every row must contain at least one positive entry.
tf_fun	A scalar aggregation function applied to the non-zero entries of each gene row. The default is mean; sum is an alternative. The function must return one finite numeric value for every row.

Details

Let x_{ij} denote entry FeatureMatrix_Gene2GoTerm[i, j]. For each gene i , the implementation first calculates the row sum

$$s_i = \sum_{j=1}^m x_{ij}$$

and the largest row sum across all genes,

$$S_{\max} = \max_{1 \leq i \leq n} s_i.$$

The inverse-document-frequency component is

$$idf_i = \log \left(1 + \frac{S_{\max}}{s_i} \right),$$

where $\log()$ denotes the natural logarithm.

Let a_i be the value returned by tf_fun when it is applied to the non-zero entries of row i , and let

$$m_i = \max_{1 \leq j \leq m} x_{ij}.$$

The term-frequency component is

$$tf_i = \frac{a_i}{m_i}.$$

The scalar TF–IDF weight for gene i is

$$w_i = tf_i idf_i.$$

Finally, the distance between genes i and j is

$$D_{ij} = |w_i - w_j|.$$

If there is a possibility that distinct genes that receive the same weight, they would have distance zero because the distance is calculated from one scalar weight per gene.

Value

A list with:

Distance	An $n \times n$ numeric matrix whose element D_{ij} is the absolute difference between the TF–IDF weights of genes i and j . Input row names, when present, are used as row and column names.
TfidfWeights	A numeric vector of length n containing one scalar TF–IDF weight per gene row. Input row names, when present, are retained.

Note

Tfidf_dist() is an alias of Tfidf_Distance() and uses the same computation and return value.

References

Stier, Q. and Thrun, M. C. (2023). Deriving homogeneous subsets from gene sets by exploiting the Gene Ontology. *Informatica*, 34(2), 357–386. doi:10.15388/23INFOR517.

See Also

[DistanceMatrix](#)

Examples

```
data(Hearingloss_N109)
V = Tfidf_Distance(Hearingloss_N109$FeatureMatrix_Gene2Term)
Distance = V$Distance
TfidfWeights = V$TfidfWeights

Distance2 = DistanceMatrix(
  Hearingloss_N109$FeatureMatrix_Gene2Term,
  method = "Tfidf_Distance"
)
```

`ToroidalEuclideanDistanceOneToAll`*Toroidal Euclidean Distances from One Position*

Description

Computes Euclidean distances from one two-dimensional position to every row of a position matrix on a rectangular torus.

Usage

```
ToroidalEuclideanDistanceOneToAll(positionxy, AllPositions, Lines, Columns)
```

Arguments

<code>positionxy</code>	A finite numerical vector $c(x, y)$.
<code>AllPositions</code>	A non-empty finite numerical matrix with two columns.
<code>Lines</code>	One positive finite extent of the first periodic coordinate.
<code>Columns</code>	One positive finite extent of the second periodic coordinate.

Details

For an absolute first-coordinate difference d_x , the wrapped difference is $\min(d_x \bmod L, L - (d_x \bmod L))$, where L is `Lines`. The second coordinate is handled analogously with `Columns`. There is no additional +1; for example, positions 1 and 40 are one grid step apart when `Lines` = 40.

Value

A numerical vector with one distance per row of `AllPositions`.

Author(s)

Michael Thrun

See Also

[ToroidalEuclidean_Distance](#)

Examples

```
ToroidalEuclideanDistanceOneToAll(  
  c(1, 1), rbind(c(40, 1), c(1, 80), c(40, 80)),  
  Lines = 40, Columns = 80  
)
```

ToroidalEuclidean_Distance

Pairwise Toroidal Euclidean Distances

Description

Computes a full pairwise Euclidean distance matrix for two-dimensional positions on a rectangular torus.

Usage

```
ToroidalEuclidean_Distance(X, Y, Lines, Columns, Points)
```

Arguments

X	A finite numerical vector of first coordinates. May be omitted when Points is supplied.
Y	A finite numerical vector of second coordinates with the same length as X. May be omitted when Points is supplied.
Lines	One positive finite extent of the first periodic coordinate.
Columns	One positive finite extent of the second periodic coordinate.
Points	Optional finite numerical matrix with two columns, used instead of X and Y.

Details

Each coordinate difference is reduced modulo its grid extent and replaced by the shorter of the direct and wrapped paths. For an extent of 40, coordinates 1 and 40 are therefore one step apart.

Value

A full symmetric numerical matrix with one row and column per position.

Author(s)

Michael Thrun

See Also

[ToroidalEuclideanDistanceOneToAll](#)

Examples

```
P = rbind(c(1, 1), c(40, 1), c(1, 80))
ToroidalEuclidean_Distance(Lines = 40, Columns = 80, Points = P)
```

TransformSimilarity2MetricDistance

Transform a Similarity Matrix to a Metric Distance Matrix

Description

Transforms a symmetric similarity matrix by $D = \sqrt{1 - S}$.

Usage

```
TransformSimilarity2MetricDistance(Similarity)
```

Arguments

Similarity	A finite symmetric positive-semidefinite square matrix with values in $[0, 1]$ and a diagonal equal to one. Under the BIDistances convention, small values mean dissimilar and large values mean similar.
------------	---

Details

Positive semidefiniteness makes *Similarity* a Gram matrix of unit vectors. Under that condition, $\sqrt{1 - S}$ is a scaled Euclidean distance. Matrices with a materially negative eigenvalue are rejected.

Value

A full symmetric numerical distance matrix with zeros on the diagonal.

Author(s)

Michael Thrun

Examples

```
S = matrix(c(1, 0.8, 0.2, 0.8, 1, 0.4, 0.2, 0.4, 1), 3, 3)
TransformSimilarity2MetricDistance(S)
```

TWED_Distance

Time Warp Edit Distance

Description

Computes Time Warp Edit Distance for two timestamped univariate or multivariate time series.

Usage

```

TWED_Distance(
  Values1, Values2, Time1, Time2,
  Nu = 1, Lambda = 1, Degree = 2
)

```

Arguments

Values1	A finite numerical vector, or a numerical matrix whose rows are time points and columns are dimensions.
Values2	A finite numerical vector or matrix with the same number of value dimensions as Values1.
Time1	A finite, positive, strictly increasing numerical vector with one timestamp per time point in Values1.
Time2	The corresponding timestamp vector for Values2.
Nu	One finite strictly positive stiffness parameter.
Lambda	One finite non-negative deletion penalty.
Degree	One finite Minkowski exponent greater than or equal to one, used for local value costs. For univariate series the local norm reduces to absolute difference; for multivariate series the exponent affects the result.

Details

A zero-valued point at timestamp zero is prepended to both series and the standard dynamic-programming recurrence is applied. Deletions combine local value change, temporal change weighted by Nu, and Lambda; a match combines current and previous value differences plus timestamp misalignment.

This distance requires the explicit time variables Time1 and Time2. It is therefore intentionally not available through DistanceMatrix(), whose integrated time-series distances use the row order and do not receive a separate time variable. Call TWED_Distance() directly.

Value

A list with components:

TWED	One non-negative numerical distance.
DPMatrix	A full numerical dynamic-programming matrix with one row per original time point of Values1 and one column per original time point of Values2.

Author(s)

Quirin Stier

References

Marteau, P.-F., Time warp edit distance with stiffness adjustment for time series matching. IEEE transactions on pattern analysis and machine intelligence, 2008. 31(2): p. 306-318.

See Also

[DTW_Distance](#), [MSMD_Distance](#)

Examples

```
x = c(1, 2, 3, 6)
y = c(1, 2, 3, 4)
result = TWED_Distance(x, y, 1:4, 1:4, Nu = 0.001, Lambda = 1)
result$TWED
stopifnot(TWED_Distance(x, x, 1:4, 1:4)$TWED == 0)
```

VariablePrecision

Smallest Within-Variable Differences

Description

Calculates the smallest absolute pairwise difference and the smallest positive absolute pairwise difference within a numerical vector or within each column of a numerical matrix.

Usage

```
VariablePrecision(Variable)
```

Arguments

Variable	A numerical vector or matrix. Non-finite values are omitted separately for each vector or column.
----------	---

Details

All pairwise signed differences are first calculated by `InnerVariableDifferences()`. Only the upper triangle is examined, so the zero diagonal of self-comparisons is excluded. Consequently, `MinAbsDiff` is zero only when two distinct observations have the same value. `MinAbsNZDiff` excludes duplicate-value differences as well.

Value

A list with components:

MinAbsDiff	The smallest absolute difference between two distinct observations, or NA when fewer than two finite observations exist.
MinAbsNZDiff	The smallest strictly positive absolute difference, or NA when none exists.
MinExpo	Currently not implemented and returned as NaN.

For matrix input, each component is a vector with one value per column.

Author(s)

Michael Thrun

Examples

```
data(Hepta)
distMat = VariablePrecision(as.matrix(iris[, 1]))

distMat = VariablePrecision(as.matrix(iris[, 1:4]))
```

Wasserstein_Distance *Wasserstein Distance Between Cases*

Description

Computes one-dimensional Wasserstein distances between all cases of a numerical matrix, treating the entries of every case as equally weighted support points.

Usage

```
Wasserstein_Distance(Data, p = 1, InverseWeighting = FALSE)
```

```
Wasserstein_Distance(Data, p = 1, InverseWeighting = FALSE)
```

Arguments

Data	[1:n,1:d] numerical matrix with n cases, d variables
p	One finite number greater than or equal to one, giving the order of the Wasserstein distance.
InverseWeighting	Logical legacy option. With FALSE, every entry receives explicit unit weight; with TRUE, every entry receives explicit weight $1/d$. Both choices define equal masses and therefore produce the same normalized transport problem.

Details

The weight vectors are always supplied explicitly to `transport::wasserstein1d()`. Each case has the same number of entries, so unit weights and weights of $1/d$ differ only by a common normalization.

Value

[1:n,1:n] numerical matrix of distances.

Author(s)

Michael Thrun

See Also

[wasserstein1d](#), [DistanceMatrix](#)

Examples

```
if (requireNamespace("transport", quietly = TRUE)) {  
  Data = rbind(c(0, 1, 2), c(1, 2, 3), c(0, 0, 4))  
  Distance1 = Wasserstein_Distance(Data)  
  Distance2 = Wasserstein_Distance(Data, InverseWeighting = TRUE)  
  stopifnot(is.matrix(Distance1), isTRUE(all.equal(Distance1, Distance2)))  
  
  data(Hepta)  
  Distance = Wasserstein_Distance(Hepta$Data)  
}
```

Index

- * **Bioinformatic Distance**
 - Cosine_Distance, [4](#)
 - Fractional_Distance, [36](#)
 - SharedNeighbor_Distance, [51](#)
- * **Bioinformatic distance**
 - Tfidf_Distance, [53](#)
- * **Cosine Distance**
 - Cosine_Distance, [4](#)
- * **Cosinus Distance**
 - Cosine_Distance, [4](#)
- * **FCPS**
 - Hepta, [39](#)
- * **Fractional Distance**
 - Fractional_Distance, [36](#)
- * **GPU computing**
 - EuclideanGPU_Distance, [28](#)
 - Minkowski_Distance, [47](#)
- * **Genes**
 - Hearingloss_N109, [38](#)
- * **Gini coefficient**
 - Gini_Distance, [37](#)
- * **GoTerm**
 - Hearingloss_N109, [38](#)
- * **Hepta**
 - Hepta, [39](#)
- * **Inverse document frequency**
 - Tfidf_Distance, [53](#)
- * **Kullback-Leibler divergence**
 - Kullback_Leibler_div, [44](#)
- * **Minkowski distance**
 - Minkowski_Distance, [47](#)
- * **Nearest neighbours**
 - DistanceOneToAll, [20](#)
- * **OpenCL**
 - EuclideanGPU_Distance, [28](#)
 - Minkowski_Distance, [47](#)
- * **Shared Neighbor Distance**
 - SharedNeighbor_Distance, [51](#)
- * **TF-IDF**
 - Tfidf_Distance, [53](#)
- * **Term frequency**
 - Tfidf_Distance, [53](#)
- * **cluster**
 - DistanceDistributionAnalysis, [6](#)
 - DistanceDistributions, [10](#)
 - IntraClusterBoundaryCounts, [40](#)
- * **concepts**
 - distance-conventions, [5](#)
 - DistanceProperties, [21](#)
- * **cosine**
 - Cosine_Distance, [4](#)
- * **datasets**
 - Hearingloss_N109, [38](#)
 - Hepta, [39](#)
- * **distance**
 - Cosine_Distance, [4](#)
 - distance-conventions, [5](#)
 - DistanceDistributionAnalysis, [6](#)
 - DistanceDistributions, [10](#)
 - DistanceOneToAll, [20](#)
 - DistanceProperties, [21](#)
 - DTW_Distance, [23](#)
 - EndresSchindelin_Distance, [24](#)
 - endresSchindelin_Distribution, [25](#)
 - endresSchindelin_TwoSamplePDF, [26](#)
 - EuclideanGPU_Distance, [28](#)
 - EuclideanMulticore_Distance, [30](#)
 - Fractional_Distance, [36](#)
 - Gini_Distance, [37](#)
 - Hellinger_Distance, [38](#)
 - IntraClusterBoundaryCounts, [40](#)
 - Jaccard_Distance, [42](#)
 - Mahalanobis_Distance, [45](#)
 - Minkowski_Distance, [47](#)
 - MSMD_Distance, [49](#)
 - NearestNeighborIndex, [50](#)
 - SharedNeighbor_Distance, [51](#)
 - Tfidf_Distance, [53](#)

- TransformSimilarity2MetricDistance, 57
- TWED_Distance, 57
- Wasserstein_Distance, 60
- * **distribution**
 - fast_dip_stat, 30
 - fast_dip_test, 32
 - fast_dip_test_matrix, 34
 - Kullback_Leibler_div, 44
- * **dtw**
 - DTW_Distance, 23
- * **high-performance computing**
 - EuclideanGPU_Distance, 28
 - Minkowski_Distance, 47
- * **htest**
 - fast_dip_stat, 30
 - fast_dip_test, 32
 - fast_dip_test_matrix, 34
- * **multivariate**
 - DistanceDistributionAnalysis, 6
 - DistanceDistributions, 10
 - fast_dip_test_matrix, 34
- * **optimal transport**
 - Wasserstein_Distance, 60
- * **probability distributions**
 - EndresSchindelin_Distance, 24
 - endresSchindelin_Distribution, 25
 - endresSchindelin_TwoSamplePDF, 26
- * **similarity**
 - TransformSimilarity2MetricDistance, 57
- * **spatial**
 - ToroidalEuclidean_Distance, 56
 - ToroidalEuclideanDistanceOneToAll, 55
- * **time series**
 - DTW_Distance, 23
 - MSMD_Distance, 49
 - TWED_Distance, 57
- * **utilities**
 - VariablePrecision, 59
- * **wasserstein distance**
 - Wasserstein_Distance, 60
- * **wasserstein**
 - Wasserstein_Distance, 60
- calculateMemoryDemandGPU, 3, 29, 48
- Cosine_Distance, 4
- dip.test.fast(fast_dip_test), 32
- dip_test_fast(fast_dip_test), 32
- dist, 23, 29
- distance, 19
- distance-conventions, 5
- DistanceDistributionAnalysis, 6, 12, 35
- DistanceDistributions, 9, 10
- DistanceMatrix, 6, 13, 22, 24, 25, 29, 36, 37, 43, 48, 50, 52, 54, 60
- DistanceOneToAll, 20
- DistanceProperties, 21
- dtw, 24
- DTW_Distance, 19, 23, 50, 59
- EndresSchindelin_Distance, 19, 24, 27
- endresSchindelin_Distribution, 6, 25, 25, 27
- endresSchindelin_TwoSamplePDF, 25, 26
- EuclideanGPU_Distance, 4, 6, 15, 28, 48
- EuclideanMulticore_Distance, 30, 48
- fast_dip_stat, 30, 34, 35
- fast_dip_test, 9, 31, 32, 35
- fast_dip_test_matrix, 9, 34, 34
- Fractional_Distance, 6, 19, 36, 48
- getDistMethods, 19
- Gini_Distance, 19, 37
- Hearingloss_N109, 38
- Hellinger_Distance, 38
- Hepta, 39
- ineq, 37
- IntraClusterBoundaryCounts, 40
- Jaccard_Distance, 19, 42
- KNearestNeighborsFromDistance, 43
- Kullback_Leibler_div, 44
- Mahalanobis_Distance, 45
- mdist, 18, 19
- Minkowski_Distance, 6, 19, 22, 47
- MSMD_Distance, 19, 24, 49, 59
- NearestNeighborIndex, 50
- parDist, 16, 19
- SharedNeighbor_Distance, 19, 51

Tfidf_dist (Tfidf_Distance), [53](#)
Tfidf_Distance, [6](#), [19](#), [53](#)
ToroidalEuclidean_Distance, [55](#), [56](#)
ToroidalEuclideanDistanceOneToAll, [55](#),
[56](#)
TransformSimilarity2MetricDistance, [57](#)
TWED_Distance, [24](#), [57](#)

VariablePrecision, [59](#)

wasserstein1d, [60](#)
Wasserstein_Distance, [19](#), [60](#)